FLIPBIT: Approximate Flash Memory for IoT Devices

Alexander Buck, Karthik Ganesan, and Natalie Enright Jerger

University of Toronto
{alexander.buck, karthik.ganesan}@mail.utoronto.ca, enright@ece.utoronto.ca

***Abstract*—IoT devices commonly use flash memory for both data and code storage. Flash memory consumes a significant portion of the overall energy of such devices. This is problematic because IoT devices are energy constrained due to their reliance on batteries or energy harvesting. To save energy, we leverage a unique property of flash memory; write operations take unequal amounts of energy depending on if we are flipping a $1 \to 0$ versus a $0 \to 1$. We exploit this asymmetry to reduce energy consumption with FLIPBIT, a hardware-software approximation approach that limits costly $0 \to 1$ transitions in flash. Instead of performing an exact write, we write an approximated value that avoids any costly $0 \to 1$ bit flips. Using FLIPBIT, we reduce the mean energy used by flash by 68% on video streaming applications while maintaining 42 dB PSNR. On machine learning models, we reduce energy by an average of 39% and up to 71% with only a 1% accuracy loss. Additionally, by reducing the number of program-erase cycles, we increase the flash lifetime by 68%.**

I. Introduction

The Internet-of-Things (IoT) spans a variety of fields such as health care, smart cities, and agriculture [2], [48], [83], [91]. In these settings, IoT devices are often powered by batteries [47], [90] or use energy harvesting to avoid being tethered to wall power [9], [30], [59], [79]. Therefore, reducing the power consumption is essential to support their widespread use.

IoT devices commonly consist of: (1) a simple CPU, (2) memory, (3) sensors to collect information (e.g., ADCs), and (4) peripherals for communication (e.g., WiFi or Bluetooth modules). The memory consists of both volatile memory (in the form of SRAM) and non-volatile memory (NVM). SRAM serves as temporary storage and includes structures such as the stack and heap, while NVM stores larger data and the program code. IoT devices typically have significantly more non-volatile than volatile memory [66], [86], [90] due to the higher density of flash compared to SRAM [14]. Thus, programs with large memory footprints often use NVM for storage when they are too big to fit into SRAM.

Many types of NVM have been proposed for IoT devices, such as FeRAM [56], STT-RAM [95] and ReRAM [52]. However, flash continues to be the most common type of NVM. For example, 67% of IoT devices in China in 2022 shipped with flash memory [31]. This dominance is due to its lower price and higher process maturity [66]. Flash memory has two varieties: NOR and NAND. NAND is used for high-density storage such as SSDs, while NOR is used for NVM in embedded devices [22], [53], [80]. We elaborate on this difference further in Section II-C.

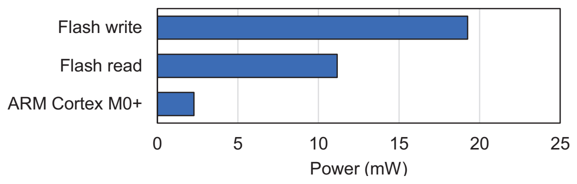


Fig. 1: Power use of different flash operations compared to an ARM Cortex M0+.

We observe that in applications with frequent writes to flash, these writes account for a significant portion of the overall energy use of IoT devices. Figure 1 shows the power consumed by read and write operations to flash memory, compared to the power used by an ARM-Cortex M0+ CPU executing ALU instructions [5], [75]; flash consumes up to $8.5\times$ more power than an ARM-Cortex M0+. Flash writes also incur high latency, leading to writes consuming **5 orders of magnitude greater energy** than reads (Section II). Thus, reducing the energy of writes can significantly reduce the total energy consumption of the device.

Another issue with flash memory is "wear out", where the device can only support a limited number (typically 10,000–1,000,000) of erase operations [11]. After this, degradation of the channel in the memory cell may cause writes to fail, leading to data corruption. IoT devices are typically deployed for long periods of time and can be challenging to replace. Thus, applications that perform many erase-program cycles can incur premature flash wear out. Aras *et al.* [4] show that without any strategies to extend lifetime, flash can wear out prior to the application's intended lifetime.

While there have been prior approaches to reduce the energy and increase lifetime of flash memory [25], [80], [98] (Section VII), we exploit a unique property of flash; changing a $0 \to 1$ requires far more energy than changing a $1 \to 0$. Flash cells – which each hold one bit – are organized in *blocks*. To write to a flash cell, the entire block must be erased, which sets all cells to 1. Then, the required cells are selectively drained to 0. If a subsequent write requires draining a $1 \to 0$, this can be done quickly and cheaply. However, to go from $0 \to 1$, the entire block must be erased and individual cells flipped to get the final value. Such *write asymmetry* requires erasing entire blocks, which incurs significant energy and latency overheads and exacerbates wear out.

2378-203X/24/$31.00 ©2024 IEEE
DOI 10.1109/HPCA57654.2024.00072

To reduce the energy of flash writes, we leverage the fact that the real-word sensor data processed by IoT devices is inherently noisy [62]. Therefore, we employ *approximate computing* to reduce the energy of flash writes for error-tolerant applications. Approximate computing has been applied to IoT devices [7], [12] and also to flash memory used in SSDs [33], [46]. However, to the best of our knowledge, we are the first to exploit write asymmetry to selectively *approximate* values written to flash.

We propose FLIPBIT, a *similarity-aware* hardware mechanism to replace expensive $0 \rightarrow 1$ writes with cheaper $1 \rightarrow 0$ writes. When performing a write, FLIPBIT compares the value already present in flash with the new value to be written. If the write only requires $1 \rightarrow 0$ flips, FLIPBIT performs the write as normal. However, if any $0 \rightarrow 1$ flips are required, FLIPBIT writes the **closest** value which does not require any $0 \rightarrow 1$ flips. Our technique does not require any changes to the CPU; we implement FLIPBIT entirely in the flash logic. We also add compiler support to annotate certain data as approximatable for our technique. Finally, to control the level of approximation, we use a programmer-specified threshold; a difference greater than this threshold causes FLIPBIT to perform a precise write, thereby limiting the error introduced by approximation.

To summarize, we make the following contributions:

- We are the first to exploit the unique property of flash memory, where flipping $0 \rightarrow 1$ costs significantly more energy and latency than flipping $1 \rightarrow 0$.
- We propose FLIPBIT, hardware to approximate values written to flash to avoid $0 \rightarrow 1$ bit-flips, while adding just 0.1% area overhead compared to an ARM M0+ SoC.
- FLIPBIT reduces energy by 39%, while incurring just a 1% drop in accuracy for a range of ML workloads.
- FLIPBIT increases the lifetime of flash memory by 68% by eliding write operations, which lead to device wear out.

II. FLASH OVERVIEW

Flash memory is a non-volatile memory commonly used in solid state drives (SSDs), USB drives, SD cards, and embedded devices. We focus on NOR memory due to its common use in IoT devices [53], [68], but review NAND memory in Section II-C. A flash cell retains information through the charge stored in a floating gate MOSFET [8]. A cell is set to 0 by injecting charge onto it through a *program* operation, and to 1 by removing the charge via an *erase* [17]. While reads and programs occur at byte granularity, the erase occurs in bulk for all cells sharing a common p-well to allow a negative voltage to appear on the control gate. While this block of cells can be up to 64 kB, some flash devices allow erase operations to occur at a smaller *page* granularity [75]; we focus on these devices with page-sized erase operations. Pages consist of all the cells that share a word line and are typically 256 or 512B.

Since an erase occurs in bulk and relies on electrons tunnelling through an oxide barrier, it is also the slowest operation [74]. In Table I, we show typical latency and energy values for the different flash operations from a commercially available flash memory used in embedded devices [75]. Critically, an *erase* operation has **340**× higher latency and **360**× higher energy consumption compared to a *program* operation. Thus, changing the logical value of a cell from $1 \rightarrow 0$ is cheaper in energy and time than going from $0 \rightarrow 1$ as the former requires only a program while the latter requires an erase.

TABLE I: Time and energy required for each flash operation.

	Read	Program	Erase
Time (ns)	30.3	30,000	10,200,000
Energy (nJ)	0.338 / Byte	545 / Byte	196,000 / 256B page

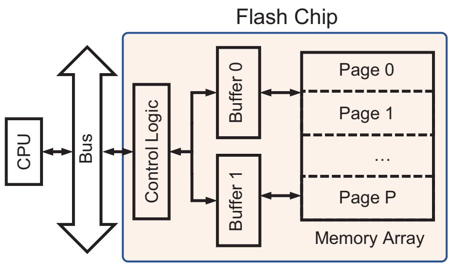


Fig. 2: Typical flash system with write buffers.

To illustrate the energy cost of flash writes in IoT, we compare it to the energy consumed by an MCU. A common embedded MCU, the ARM Cortex-M0+, consumes 2.275 mW running at a typical 48 MHz in 180 nm technology [5]. During the 10.2 ms required to perform a page erase, the MCU consumes 23.2 μJ. Meanwhile, during the erase, flash consumes 196 μJ or 8.4× as much energy as the MCU.

A. Flash Organization

Figure 2 illustrates the typical organization of flash. The flash memory array is split into several pages. The CPU and the flash are connected via a bus. The control logic in the flash communicates with I/O, decodes addresses, and controls all flash operations. The memory array is connected to the control logic through write buffers, which we describe next.

Write Buffers: To increase write speed and reduce flash energy, manufacturers typically add volatile SRAM write buffers to flash chips [38], [75]. These are used to quickly program entire pages of cells. In addition to reducing write energy, write buffers allow a "read-modify-write" operation in which an entire page is erased and then programmed in one command [75]. This operation works in 4 steps: (1) The entire page is read into the buffer, (2) the page in the flash is erased, (3) the values are modified by the CPU in the buffer, and (4) the CPU sends a command to write back the entire buffer to the flash array. This operation allows any sequential block of bytes within a page to be modified without changing the rest of the page. While a single buffer allows for faster access to a single page, many designs provide two buffers to interleave streams for operations that edit multiple pages in quick succession [76]. FLIPBIT leverages these write buffers to achieve energy reduction.

B. Endurance

An important consideration for flash memory is that it has a limited number of program-erase cycles. This is mainly due to tunnel oxide degradation as charges get trapped in the oxide [8]. The number of these cycles varies by manufacturer, ranging from 10,000–1,000,000 cycles [11]. Therefore, various techniques – such as using a flash translation layer (FTL) [19] or flash file system [26], [94] – have been proposed to achieve better wear levelling. These techniques aim to distribute writes across different pages through address translation to prevent one page from failing earlier than the rest of the memory array. However, they either require an operating system or a dedicated MCU on the flash chip itself to manage these operations. Additionally, there are performance and memory overheads to manage the address translations.

Since IoT devices are typically small, the memory and performance overheads associated with using an FTL or file system can be prohibitive [13], [25]. In these cases, the onus is on the programmer to prevent early wear out by distributing writes evenly across the memory [37], [87]. Since the lifetime of the flash device depends on the number of program-erase cycles, we describe in Section III-B how our technique, FLIPBIT, increases lifetime by directly reducing the number of page erases. Therefore, FLIPBIT extends flash lifetime without the additional overhead of a flash translation layer or file system. Thus, our technique is orthogonal to existing wear levelling techniques and can be easily combined with them to yield further benefits.

C. Types of Flash Memory

Flash memory varies based on the structure of the transistor used and the number of bits stored per cell [15]. We elaborate on these differences in the context of IoT devices, which is the focus of our work.

NOR vs. NAND: Flash memory is categorized as either NOR or NAND flash, based on the structure of transistors used to store charge. NAND flash has higher density and is typically used in large storage mediums such as SSDs [15]. However, NOR is preferred for embedded devices for several reasons: (1) NOR supports random read access, while NAND only supports sequential reads. As NOR flash is used for storing instructions, efficient random reads are essential for good performance. (2) NOR supports eXecute-in-Place (XIP), which allows code to be run directly from flash memory without having to be copied to SRAM first [28]. XIP reduces the total memory requirement which is important in a constrained IoT environment. (3) NOR is more reliable than NAND and so does not require the added overhead of error checking codes (ECC) [36]. For these reasons, NOR flash is commonly used in IoT devices for both storage and code execution and therefore is the focus of our work [22], [28], [53], [68], [80].

SLC vs. MLC: Flash varies in the number of bits stored per cell. Single-level cells (SLC) store one bit per cell while multi-level cells (MLC) store multiple bits per cell. MLC still requires an erase to remove charge from the cell while a program only adds charge to the floating gate. SLC has higher endurance, reliability, and read speed compared to MLC [88]. While MLC is common in high density NAND flash, SLC is common in embedded devices requiring code execution for its faster latency and higher reliability [3], [88]. Although we focus on SLC flash, in Section VI we discuss how FLIPBIT can be applied to MLC flash.

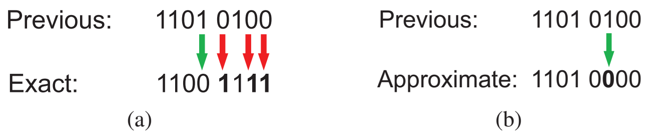


Fig. 3: Example of using FLIPBIT to perform an approximate store. (a) For the exact write, we require expensive 0 to 1 transitions (red arrows). (b) For the approximate write, we only use inexpensive 1 to 0 writes (green arrows).

Having established the key properties of flash storage, we next describe FLIPBIT, which reduces energy and increase lifetime of NOR flash in IoT devices.

III. FLIPBIT

As described in Section II, changing a cell from $1 \rightarrow 0$ is cheaper than going from $0 \rightarrow 1$. FLIPBIT leverages this technique to *approximate* writes to flash memory to reduce energy. We selectively write a 0 to a 1 while avoiding writing 1s to 0s. Suppose that a memory location has previously stored $1101\,0100_2$ (212_{10}) and we want to store $1100\,1111_2$ (207_{10}). As we show in Figure 3a, this would require changing several 0s to 1s and trigger a costly erase operation. Instead, FLIPBIT avoids erases by approximating this value as $1101\,0000_2$ (208_{10}) with an error of only 0.48% (Figure 3b).

A. Approximation Algorithms

Our goal is to design a *hardware-friendly algorithm* to generate the approximate values. In software, we can take a sequential approach; however, such a design would add complexity in hardware due to branches. In contrast, we opt for a design which works in parallel and therefore, is better suited for hardware.

We now describe our baseline approximation algorithm.

1) Baseline Approximation Algorithm: The baseline algorithm always returns the value closest to the exact value without flipping any 0s to 1s. Formally, we want to find *approx* that minimizes $|exact - approx|$ such that for every bit, *i*, in *approx[i]* that equals 1, *previous[i]* also equals 1.

$$\begin{aligned} &\min_{approx} \mid exact - approx \mid \\ &\text{s.t.} \quad previous[i] \;=\; 1 \quad \forall i \in approx[i] \;=\; 1 \end{aligned} \tag{1}$$

While the baseline algorithm minimizes error between exact and approximate values, it is also complex to implement. Effectively, we have to generate every possible combination in which we either flip a 1 into a 0, or keep it as a 1 and then compare this value with the exact one. To see the issue with scaling, consider a value with n number of bits as 1. Then, the number of possible new values we could generate is 2^n. For example, if our original binary value is 101, then we could generate $101, 100, 001$, and 000.

Consider the simple case of random data, where half the bits are 1 and other half are 0. Thus, for an 8 bit value, on

Algorithm 1 Approximation by examining one bit at a time

```
Input: previous, exact        ▷ Where variables have $n$ bits
Output: approx
 1: approx ← 0
 2: setOnes ← false
 3: for $i \leftarrow n-1 \ldots 0$ do
 4:   if previous[$i$] = 1 then
 5:     if exact[$i$] = 1 or setOnes = true then
 6:       approx[$i$] ← 1
 7:     end if
 8:   else if exact[$i$] = 1 then
 9:     setOnes ← true
10:   end if
11: end for
```

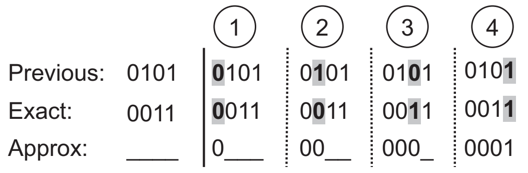


Fig. 4: Creating an approximate value using Algorithm 1.

average, n = 4, resulting in $2^4 = 16$ possible new values. While comparing between 16 values may be doable, the exponential growth quickly makes this strategy unfeasible. For example, 16-bit and 32-bit values would on average result in $2^8 = 256$ and $2^{16} = 65,536$ new values to compare.

To combat the issue of poor scalability, we explore alternative algorithms to generate the approximate value. The key insight we rely on is that the most significant bits are much more important than the least significant bits to get a reasonable approximate value. That is, we can make a decision about bit n without looking at all bits $n-1, n-2, n-3, \ldots, 0$. Instead we can focus on a smaller subset such as $n-1$ and $n-2$. In the simplest case, we could make a decision looking solely at bit n. We now explain how this would work for n = 1.

2) One-Bit Approximation Algorithm: The one-bit approximation algorithm goes through each bit starting from the most significant bit (MSB) to determine if we should set the approximate output bit to 0 or 1. If the previous bit is 1, then we set the output to 1 in two cases; (1) if either the exact bit is 1 or (2) we already encountered an exact bit that was 1 but the corresponding previous bit was 0. In the second case, we were not able to set the output bit to 1 since this would require an erase operation. This means that the final value will be smaller than the exact, so we minimize the error by setting all following bits to 1. We detail this approach in Algorithm 1 and provide an example below.

Algorithm 1 takes 2 inputs, the previous value we are overwriting (*previous*) and the exact value we want to write (*exact*). We output *approx*, the approximated value that does not require any erase operations. For example, given *previous* = 0101 and *exact* = 0011, we want to determine *approx*. Figure 4 shows how the *approx* is built bit by bit. In line 3, we start by looking at the MSB of *previous* and *exact*, i.e., *previous*[3] and *exact*[3] (step ① of Figure 4). Since both bits are 0, we keep *approx*[3] as 0 and proceed to the next iteration.

In the next iteration of the for loop (step ② of Figure 4), we find that *previous*[2] = 1, but since *exact*[2] = 0, we proceed to the next iteration. Here *previous*[1] = 0, but *exact*[1] = 1 (step ③ of Figure 4). This means that the approximate value will be strictly less than the exact value. Thus, to minimize the error, we want to make *approx* as large as possible with the bits that are left by setting them all to 1. We achieve this by setting the *setOnes* signal to true in line 9. In the final iteration (step ④), since *previous*[0] = 1 and *exact*[0] = 1, we set *approx*[0] = 1. This gives us a final result of *approx* = 0001.

The final approximation in the example has an absolute error of 2_{10}. The baseline algorithm would have yielded *approx* = 0100 giving an error of 1_{10}. However, by examining one bit at a time, we reduced the complexity to $O(n)$ operations, where n is the number of bits in the value. Using the baseline algorithm would have required $O(2^m)$ operations, where m is the number of 1s in the *previous* value.

We now have two strategies to determine an approximate value, the baseline that minimizes error, but suffers from high implementation costs, and the 1-bit approximation that is simpler to implement, but can suffer from bigger errors. As a next step, we look for a happy medium between these strategies that has feasible implementation costs, but achieves a smaller error than the 1-bit approximation. We do this by looking at several bits at once in a method we call “n-bit approximation”.

3) N-Bit Approximation Algorithm: The n-bit approximation algorithm is a generalization of the 1-bit approximation method that examines more than one bit at a time to determine the value for *approx*[i]. In Table II, we show our derived truth table for n-bit approximation where n = 2. The first two rows of the table are the same as the 1-bit approximation method. We precompute the other rows by adopting a strategy of minimizing the maximum potential error.

To derive our truth table, we determine the minimum and maximum possible values that *exact* could be given the information we have. For example, if *exact*[n:1] = 0, then the maximum value *exact* could be is 1 if *exact*[0] = 1. Inversely, the minimum value would be 0 if *exact*[0] = 0. Once we know the minimum and maximum possible values for each combination of *exact*, we determine what the maximum error would be. Determining the maximum error requires using information from *previous*. For example, if *previous*[1:0] = 11, then we know that we can always set bits [1:0] to 0 or 1. However, if *previous*[1:0] = 01, then we know only bit 0 can be set to 0 or 1, while bit 1 must be 0. Using this information lets us determine the maximum potential error if we set *approx*[i] to 0 or 1. We then set *approx*[i] to a value such that it minimizes the maximum potential error.

We performed the procedure described above for all rows in the truth table and for values of n from 2–8. We generalize Algorithm 1 to work with the n-bit approximation. Algorithm 2 shows how to use the above truth table to generate an approximate value. In the case where we set *approx*[i] = 1 even

TABLE II: Truth table for n-bit approximation where n = 2

exact[i]	exact[i-1]	previous[i]	previous[i-1]	approx[i]
x	x	0	x	0
1	x	1	x	1
0	0	1	0	0
0	0	1	1	0
0	1	1	0	1
0	1	1	1	0

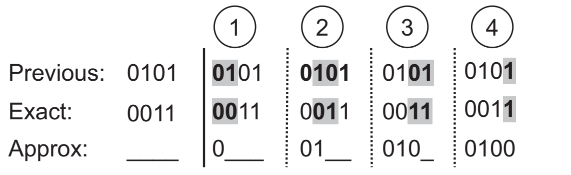


Fig. 5: Creating an approximate value using Algorithm 2 when n = 2.

when *exact*[i] = 0, we know our *approx* value will be greater than *exact*. In this case, we want to prevent any less significant bits being set to 1 to minimize the error. Accordingly, we add a new flag, *setZeros*, to achieve this purpose. It gets set in line 11 only if *approx*[i] = 1, but *exact*[i] = 0.

In Figure 5, we show how an approximate value is created using n-bit approximation for n = 2 using the same example *previous* and *exact* values as used in Figure 4. Similar to the 1-bit approximation, in step ①, the output remains 0. However, in step ②, we now set *approx*[2] = 1 since *exact*[2:1] = 01. The rest of the bits then become 0 resulting in a final *approx* = 0100. Compared to the 1-bit algorithm where *approx* = 0001, we have reduced the error to the *exact* value from 2 to 1. Having explored different strategies to generate approximations, we now look at how we can bound the error produced by the approximation.

4) Error Tolerance: When using approximate computing, it is important to control how much error may be introduced in the application. We achieve this by tracking the absolute error between the exact and approximate writes. When we perform multiple approximations across an entire flash page, we use the mean absolute error (MAE) for all the values that are written. We use MAE as it requires less hardware to implement than the commonly used mean squared error. Once we have the MAE, we use this information to make a decision about the approximation. We use a programmer-specified threshold to determine if we should write the exact value (requiring an erase) or perform the approximation. This threshold can be tuned on a per-application basis.

B. Hardware Implementations

We implement the hardware to perform the n-bit approximation and to track the MAE. Figure 6 implements one iteration of Algorithm 2 in hardware. It outputs a single bit of *approx*. It takes as inputs n bits of *previous* and *exact* and the *setOnes* and *setZeros* signals. The various if statements in the algorithm can be implemented through multiplexers. However, we found that many of the multiplexer inputs were tied to 0 or 1, so we further simplified the logic in our final design.

Algorithm 2 N-bit approximation algorithm

```
Input: $previous$, $exact$          ▷ Where variables have $n$ bits
Output: $approx$
 1: $approx \leftarrow 0$
 2: setOnes ← false
 3: setZeros ← false
 4: for $i \leftarrow n-1 \ldots 0$ do
 5:   if setZeros = false then
 6:     if $previous[i] = 1$ then
 7:       if $exact[i] = 1$ or setOnes = true then
 8:         $approx[i] \leftarrow 1$
 9:       else if TRUTHTABLEVALUE($exact$, $previous$, $i$) then
10:         $approx[i] \leftarrow 1$
11:         setZeros ← true
12:       end if
13:     else if $exact[i] = 1$ then
14:       setOnes ← true
15:     end if
16:   end if
17: end for
```

The *truth table logic* block (used in Line 9 of Algorithm 2) implements Table II for different values of n through combinational logic. If we want to use different values of n for the n-bit approximation, one option is to have a different table for each value of n. Alternatively, due to the regularity of the truth table construction, if we implement the truth table for some n_{max}, it also contains the truth table for all values of $n < n_{max}$. Specifically, by tying the m least significant *exact* and *previous* inputs to 0, we create the truth table for $n_{max} - m$. In our evaluation we consider an n_{max} of 8. Thus, we use a single circuit for all 1, 2, …, 8-bit approximation algorithms that is configurable at run-time.

The circuit in Figure 6 is duplicated 32 times so it can be used to generate the approximation for a 32 bit integer. We show how the duplicated circuits connect in Figure 7. The *setOnes* and *setZeros* signals are propagated through the design from blocks $31 \rightarrow 30 \rightarrow \cdots \rightarrow 0$. Each additionally takes n values of both the *previous* and *exact* signals. Blocks where $i < n$ have *previous* and *exact* zero padded since in this case our inputs would index from bits < 0. If a variable width is less than 32 bits, i.e., 16 or 8 bits, we only use the lower 16 or 8 blocks to generate the approximate value. In Section V-D, we synthesize our design and discuss its minimal overhead.

System Integration: Since many flash chips have write buffers, we add our approximation between the write buffer and the memory array [38], [75], [76]. Figure 8 shows the proposed modifications in the flash chip. For our design, we use two buffers so that we can keep a copy of the exact and approximate version simultaneously. This lets us choose which version to write to flash depending on how much error the approximate version introduces. While the second buffer can be used to interleave streams when needing to modify multiple pages at

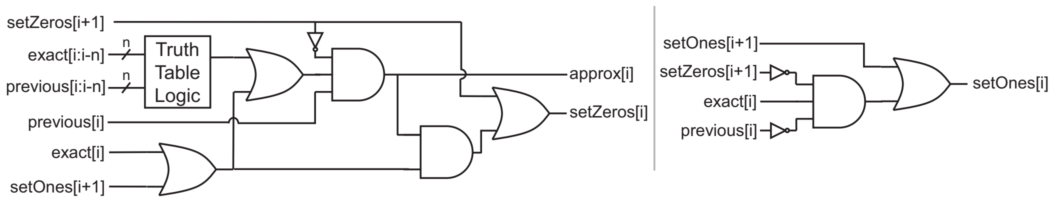


Fig. 6: The n-bit approximation circuit for bit i.

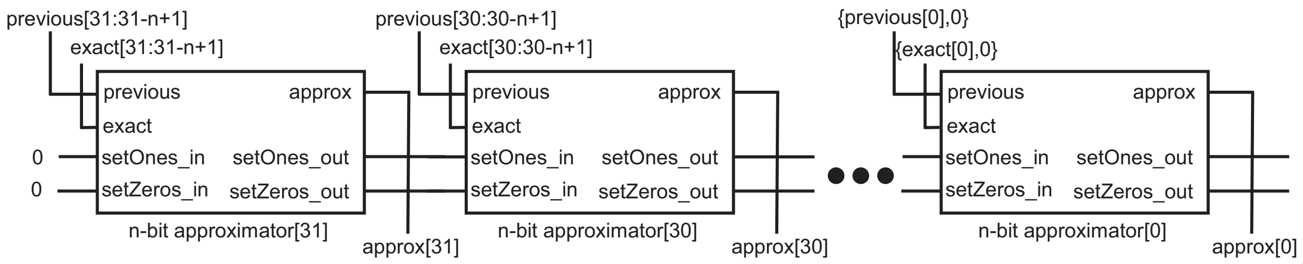


Fig. 7: The circuit to generate a 32-bit value using 32 copies of the n-bit approximator hardware.

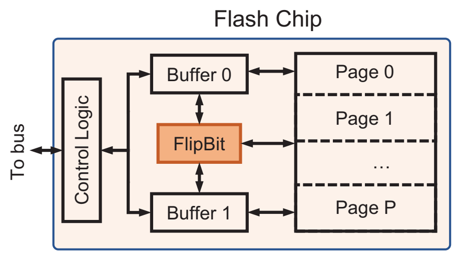


Fig. 8: Overview of the proposed flash system, showing FLIPBIT added to the baseline configuration.

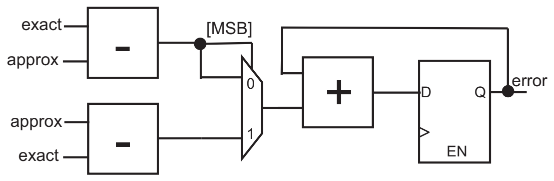


Fig. 9: The circuit to track the MAE.

once, in the applications we studied, memory writes were mostly sequential so we only needed one write buffer to maintain performance. This leaves one buffer free to be used for our approximation strategy. Approximation can be disabled in cases that require two buffers for higher write bandwidth.

To use FLIPBIT, we employ a similar mechanism to the "read-write-modify" operation. Recall from Section II that a read-write-modify first reads the flash page into a buffer. However, when using FLIPBIT, we read the page into both buffers 0 and 1. Unlike a read-write-modify, we do not automatically erase the flash page. Instead, the CPU writes the exact values to buffer 0. Once the CPU has completed writing all values to that page, we modify the stored values in buffer 1 using our approximation hardware. We want to ensure the approximation does not produce too much error, so we have to compare our approximate with exact values.

We use the circuit in Figure 9 to measure the MAE between buffer 0 (exact) and buffer 1 (approximated). It computes the absolute difference between the exact and approximate value and accumulates this error for each value in the page. If the error is less than the programmer supplied threshold, we program the flash from buffer 1. Otherwise, we have to use buffer 0 for an exact write to an erased page.

C. System Interface

We now describe how our hardware interfaces with the CPU and software. FLIPBIT needs to know:

1) the memory region that is approximatable,
2) the width of the variable type, and
3) the MAE threshold to know how much error is allowed.

Flash already uses memory-mapped registers for configuring and reading values such as write-protection settings and flash status [87]. We extend the use of memory-mapped registers for our application. Specifically, we require 4 registers, two to store the start and end address of the approximatable memory region, one for the variable type, and one for the MAE threshold. To change the value in these registers, we perform a *store* operation from the CPU to the memory address of that register. Then during run-time, when a value is to be written to the flash memory array, we check whether the address is within the set range for approximation. Since the hardware needs to know whether it should run for an 8, 16, or 32-bit variable when generating the approximate value and computing the error, we also read this information from the register. Similarly, the threshold is read from the register to determine whether we exceed the allowable error.

Software: We expose the approximation interface to the programmer through an extension to the compiler and a library function to set the MAE threshold. Similar to prior work [81], we add an `approx` keyword to annotate approximatable variables. Any variable with the `approx` keyword is stored in the approximate region of memory. The type width is also

Listing 1: Sample C program calculating the output of one fully connected layer using FLIPBIT. Lines highlighted in gray show modifications required by the programmer.

```
uint8_t weights[IN_SIZE][OUT_SIZE];
uint8_t bias_weights[IN_SIZE][OUT_SIZE];
approx uint8_t out[OUT_SIZE];

setApproxThreshold(2);
for (int i = 0; i < OUT_SIZE; i++){
  uint8_t acum = 0;
  for (int j = 0; j < IN_SIZE; j++){
    acum += input[j] * weights[i][j];
  }
  out[i] = relu(acum + bias_weights[i]);
}
```

Listing 2: Sample linking script.

```
MEMORY
{
  ram    : ORIGIN = 0x10000, LENGTH = 4K
  rom    : ORIGIN = 0x20000, LENGTH = 64K
  approx : ORIGIN = 0x40000, LENGTH = 1M
}
```

communicated to the flash through code added by the compiler for variables marked by `approx`.

In Listing 1, we show an example of using FLIPBIT to calculate the output of one fully connected layer of a neural network. The weights are declared as regular `uint8_t` arrays while we add the `approx` keyword only to the activation output (`out`). Therefore, only `out` will be stored in the approximate region of memory while all other variables are in the exact region. When `out` is updated, it will use approximation, while other variables do not.

Linking: Using our approach, the linker needs to know where to physically put the approximate region in memory through a linker script. In embedded systems, a linker script is used to manually manage the location and size of memory regions. We show a snippet of an example linker script in Listing 2 which specifies the physical address of varying memory regions. When using FLIPBIT, an additional "approx" region is specified that matches the approximatable memory region in the flash. Consequently, all variables marked by the `approx` keyword are stored in the "approx" region specified by the linker script. To communicate this region to the flash, the compiler inserts code to set the memory-mapped register with the appropriate start and end addresses.

To set the MAE threshold, we construct a `setApproxThreshold(unsigned thresh)` function. Using this function, the programmer can adjust the threshold in software. They may also adjust the threshold dynamically in the program. Internally, this is achieved through changing the value in the memory-mapped register.

Having established the various components of the system interface, we now describe how the system would be used. As shown by the highlighted lines in Listing 1, the programmer starts by marking certain variables in the program as *approx* and setting a threshold with the `setApproxThreshold()` function. Next, the programmer compiles the program and sets the size of the approximate region. When the compiled program is run on the CPU, the start and end addresses of the approximate region, variable width, and threshold are loaded to the flash through memory-mapped registers. Since the flash hardware now has all required information, the program will run in its approximate form.

IV. METHODOLOGY

To evaluate FLIPBIT, we modify a cycle accurate ARM Cortex-M0+ simulator [35] to collect statistics and perform approximation in the flash. We multiply the number of times different flash operations (read, program, erase) occurred by the energy consumed for those operations using data from a commercially available flash chip [75]. The ARM Cortex-M0+ CPU does not have a cache [5] which is typical for low-power IoT devices [66]. Larger IoT devices, such as the Samsung ARTIK 053/053s [84], may have a cache, but their caches are still too small (32 kB) to fit our workloads. The applications we evaluate have large memory footprints (100s of kB). Thus, the working set does not fit in the cache or SRAM, and still requires writes to flash to run. We focus on integer and fixed point data in our evaluation, matching prior work which runs machine learning inference on embedded devices [29], [40], [71], [99]. In addition, lower power MCUs do not have floating-point units [5]. We discuss extending FLIPBIT to floating-point numbers in Section VI.

IoT devices typically operate in one of two paradigms: 1) sense and send data and 2) compute and send data [21]. To demonstrate the broad applicability of FLIPBIT, we evaluate applications from both paradigms. For **sense and send**, we examine capturing video from a camera. IoT devices are frequently used for video capture in scenarios such as wildlife tracking [23], [101], traffic monitoring [97], and agricultural crop monitoring [51]. Video capture requires saving an image from the camera to the device before it is transmitted off the device for processing. Due to the small amount of SRAM available on IoT devices, these images must be written to flash before they can be transmitted. We apply approximation to the flash location that is repeatedly written to. We use videos from the open source Xiph.org Video Test Media collection as our benchmark suite [96]. To measure the quality of the final video, we use peak signal-to-noise ratio (PSNR) between each approximated and exact frame. We then take the average for all frames to get our final PSNR value.

For **compute and send**, we evaluate deep neural networks (DNNs) which are commonly used to process data before transmission [1], [49], [93]. A key challenge of running DNNs on constrained IoT devices is the small amount of memory available. Due to the high memory footprint of DNNs, we write the activation output of each layer to flash; we apply

TABLE III: ML models evaluated.

Model	Application	Name	Parameters	Model Size (kB)
CNN	Image Classification	mnist_cnn	3,620	7.07
MLP	Image Classification	mnist_mlp	101,770	199
CNN	Human Activity Recognition	har_cnn	738,950	1,443
MLP	ECG Abnormal Heartbeat Detection	ecg_mlp	37,801	73.8

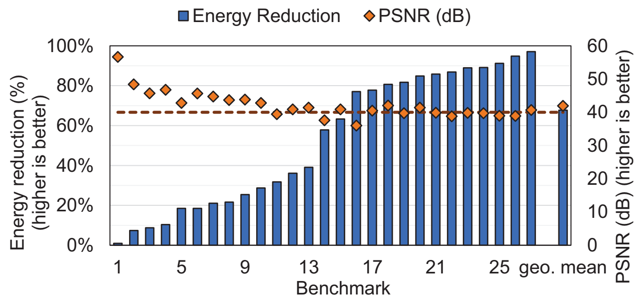


Fig. 10: Energy reduction and accuracy using 2-bit approximation for video benchmarks. The line at 40 dB represents a level of quality above which humans do not perceive a difference in images [16], [41].

our approximation to the those activation outputs prior to writing them to flash. We test four different DNNs run on three applications and summarized in Table III. We evaluate image classification, a common IoT application [69], on the MNIST dataset [42] using a convolutional neural network (CNN) and multilayer perceptron (MLP). We use another CNN for human activity recognition (HAR) [32]. Finally, we use an MLP for detecting an abnormal heartbeat in an ECG [54].

V. Evaluation

FlipBit provides energy savings with minimal and tunable quality degradation. We start by evaluating the 2-bit approximation approach; we explore different values of n for n-bit approximation in Section V-B. In Figure 10, we evaluate the energy reduction and approximation quality of the 2-bit approximation approach on the Xiph.org video suite. We sort the benchmarks according to the energy reduction we see and assign each one a unique ID. We set a threshold of 5 for all videos to minimize the quality loss. We perform a sweep of thresholds in Section V-A.

As seen in Figure 10, the energy reduction varies significantly across the videos we test. We see flash memory energy reductions ranging from 1.0%–97.1% with a geometric mean of 67.7%. The approximation hardware itself adds less than 0.02% additional energy. The videos with greater energy reduction (>80%) have overall stability; they have fixed cameras with little background movement and only objects moving in the foreground. More dynamic videos saw smaller energy reductions (<10%). They are recorded using handheld cameras and have more action occurring in the fore and background. For example, video 1 consists of a shaky, handheld camera filming a husky running near a river. We expect FlipBit to be used in mainly low-energy IoT applications. In these, capturing stable images, such as from a mounted wildlife camera, is more common than capturing highly dynamic scenes with a handheld device. Thus, we expect FlipBit to perform similarly to the more stable videos in IoT applications with significant energy reductions.

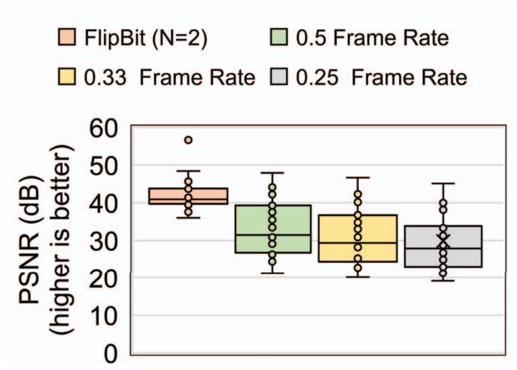


Fig. 11: Comparing PSNR for 2-bit approximation to reducing the frame rate.

While energy reduction varied significantly, video quality saw less variation. In Figure 10, we also show the PSNR for each video. As variations in image quality at or above 40 dB are considered nearly indistinguishable by humans and acceptable for medical imaging [16], [41], we add a line to Figure 10 to indicate the acceptable quality level. We find that across videos, we obtain a geometric mean PSNR of 41.9 dB. Only video 1 (husky) has a much higher PSNR. This is due to the dynamic nature of the video and therefore difficulty in applying approximation. Thus, FlipBit avoids almost all approximation to maintain output quality.

Reducing Frame Rate: A more straightforward approach rather than adding hardware to approximate values would be to reduce the frame rate, thus necessitating fewer writes to flash. Specifically, the energy consumed is directly proportional to the frame rate. That is, when reducing the frame rate by $n\%$, the energy consumed is $n\%$ of the original. This approach can yield strong energy savings, but has a significant cost to output quality. As shown in Figure 11, the 2-bit approximation technique has a higher average PSNR compared to statically reducing the frame rate. This is because our technique takes into account the error introduced through our threshold. Thus, videos with considerable movement use less approximation in the dynamic parts of the frame, resulting in less quality loss. Conversely, the loss in output quality when reducing the frame rate has a much higher correlation to the amount of movement in the video because a frame may have many changes by the time it is next updated. This results in the lower average PSNR compared to our technique.

Additionally, the distribution of output quality degradation of FlipBit is smaller compared to reducing the frame rate. Again, this is because we take into account the error that is introduced by applying our approximation. Accordingly, FlipBit can be

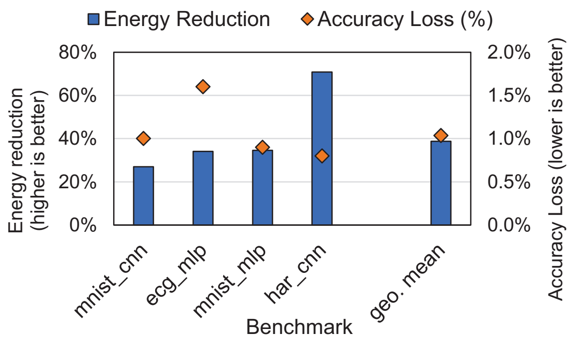


Fig. 12: Energy reduction and accuracy loss compared to precise execution using 2-bit approximation for ML benchmarks.

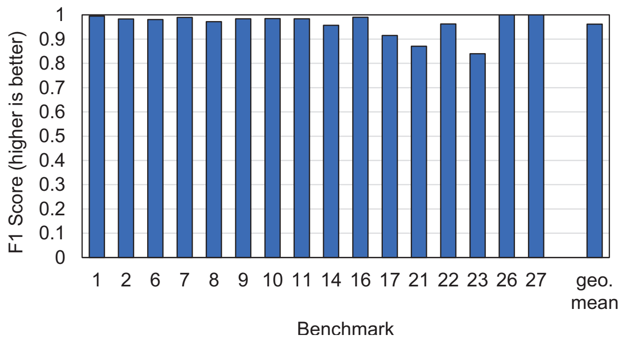


Fig. 13: F1 score of 2-bit approximation.

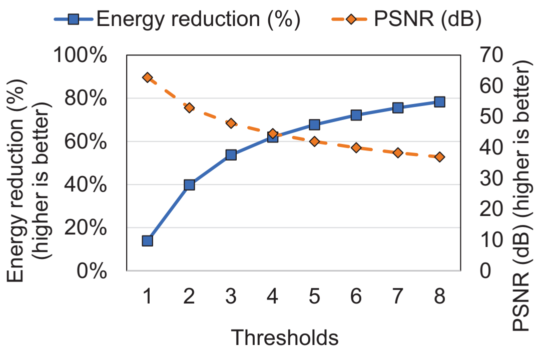


Fig. 14: Geo. mean of energy and PSNR for all videos while sweeping the threshold in the 2-bit approximation.

used on a wider range of videos as it will reduce approximation when it finds the video not amenable to such techniques.

ML Benchmarks: In Figure 12, we evaluate FLIPBIT on the 4 ML benchmarks from Table III. We continue to use 2-bit approximation and choose a threshold for each model to keep the accuracy reduction close to 1%. We will show sweeps of this threshold in Section V-A. The energy reduction has a geometric mean of 39% while the accuracy drops 1.04%. The HAR CNN has the highest energy reduction. This is because HAR CNN is the largest model and therefore has the most writes to flash as each layer's output is propagated through the model.

End-to-End System: While PSNR is a frequently used metric for image and video quality, it often cannot represent the final quality of applications that use images. Therefore, to perform an end-to-end evaluation, we consider IoT devices doing wildlife capture in which the end goal is to detect the presence of certain animals. We evaluate the performance of object detection on the approximated videos. We use the YOLOv3 model run on the Python ImageAI library [67], [73]. We compare the objects detected by the approximate versus the exact, baseline version. To determine if a predicted object matches the baseline, we use Intersection over Union (IoU) with a threshold of 50%, similar to prior work [50]. IoU measures the overlap of the predicted box surrounding an object with the actual box surrounding the object. Then we use the precision and recall to determine an F1 score.[1] Because not all our input videos have objects that YOLOv3 could properly detect, we remove the videos where the model could not clearly detect objects in the baseline.

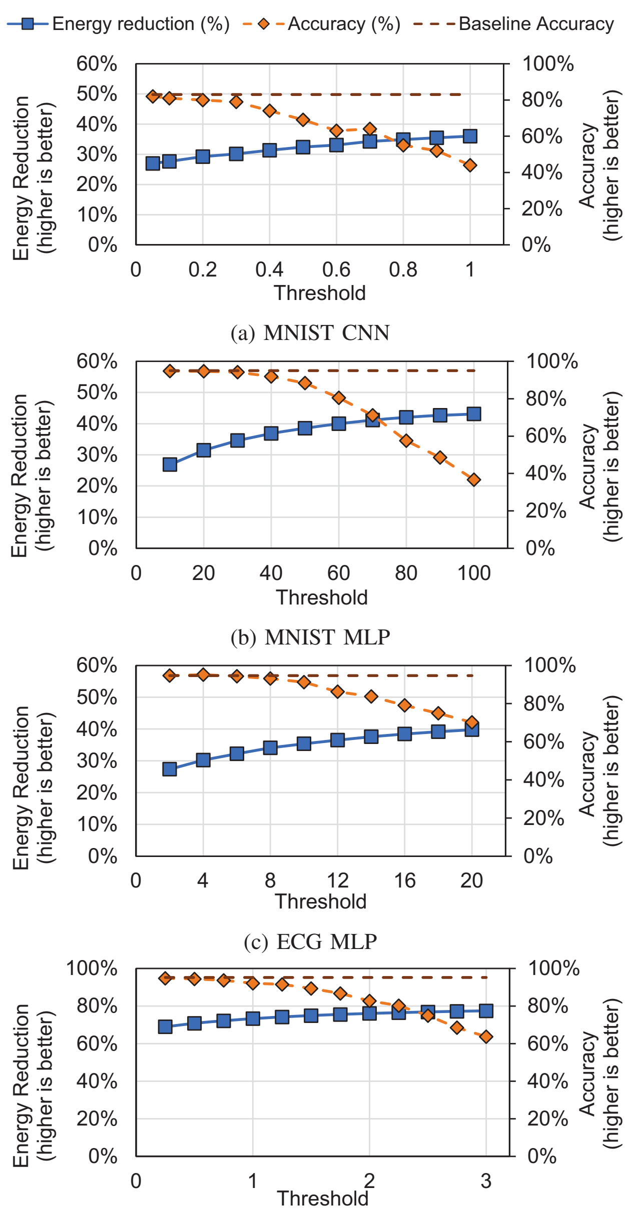
In Figure 13, we show the final performance of object detection when the input video stream has been approximated. With a geometric mean F1 score of 0.96, we find that FLIPBIT does not negatively the affect the end-to-end performance by an appreciable amount. Recall from Figure 10 that videos with lower IDs had less energy reduction and generally higher PSNR. Therefore, these videos tend to perform very well in object detection. The type of video also had an effect on the resulting F1 score. For example, 26 and 27 both consist of a person talking to the camera while the poorer performing 23 tracks a boat moving on water. Detecting the boat was more difficult due to more movement in the background and a higher number of objects in the frame.

[1]Precision $= \frac{TP}{TP+FP}$, Recall $= \frac{TP}{TP+FN}$, F1 $= 2\frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$ where TP = True Positive, FP = False Positive, and FN = False Negative

A. Sweeping Thresholds

Having previously looked at a single threshold, we now sweep the threshold to show how it affects energy reduction at a cost of quality loss. In Figure 14, we first perform a sweep of the thresholds for the video applications. As the thresholds increase, our energy savings increase and PSNR decreases. Since the threshold is controlled by the user through software, this provides an easy way to configure FLIPBIT for different tradeoffs. Thus, depending on the needs of the application, the quality loss can be set through the threshold such that it meets the requirements of the final output.

As the thresholds continue to increase, the energy reduction starts to level off. This occurs because values that are easy to approximate become more rare to find. For example, parts of the video that are very static are easier to approximate than the dynamic parts. Therefore, the error is very high for the dynamic sections. Thus, with a low threshold, only parts of the video which are very static get approximated. However, even with a higher threshold, approximating dynamic parts of the video still causes such a high error that it triggers an exact write, which does not yield any energy savings.



(a) MNIST CNN

(b) MNIST MLP

(c) ECG MLP

(d) HAR CNN

Fig. 15: Evaluation of DNNs when sweeping thresholds.

Next, we evaluate sweeping thresholds for our ML applications. Unlike the video benchmarks, we find that the ideal threshold varies per network. To automate the process of finding the threshold, we start with a value of 0.1 and multiply by 10 each time. Thus, by running just four threshold values (0.1, 1, 10 and 100), we can quickly narrow down the threshold range. We then sweep through values within this range to determine the ideal threshold per network. As we do this sweeping in software, we can quickly identify the final threshold value with minimal programmer effort. This is similar to existing approximate computing techniques which also rely on setting pre-determined thresholds [18], [78], [85], [92], [100].

In Figure 15, we show the energy reduction and accuracy loss for the four networks while sweeping the threshold. Similar to the videos, as the threshold increases, we save more energy but at the cost of greater accuracy loss. However, in the DNNs, the energy reduction is less steep than those observed for videos in Figure 14. We believe this is because most of the energy savings come from times when the activation outputs are set to zero. This occurs frequently in DNNs due to the ReLu function. We observed that between inference iterations, outputs frequently get set to zero which benefits our technique since setting a value to zero can be achieved perfectly without requiring an erase. Thus, when we set higher thresholds, the approximations we perform are not close to the exact value and results in more error. This leads to fewer pages being written without an erase and therefore less energy savings.

B. *Different Approximation Schemes*

Next, we sweep the number of bits, N, in the N-bit algorithm. In Figure 16, we show the results for the video benchmarks. We find that once we use 2 or more bits, we get nearly uniform energy savings. This is because of the exponential dropoff in importance of less significant bits compared to more significant bits. Additionally, since we check that the error within a page is not too high, even a more precise approximation may still result in requiring an erase.

C. *Lifetime and Endurance*

In addition to reducing energy used by flash, FLIPBIT also increases the lifetime of the flash. IoT devices are often deployed in remote areas, which can make the replacement of parts difficult. Therefore, increasing the lifetime of flash is important for IoT devices.

To evaluate the benefit in endurance FLIPBIT provides, we compare the number of pages requiring an erase using approximation and not. Since flash endurance relies on the number of program-erase cycles, we use a reduction in flash page erases as a proxy for lifetime increase. This proxy is valid if the most frequently written to pages are also those being approximated. In this case, those pages would normally wear out soonest, so reducing the number of erases for the affected pages will also increase the overall lifetime. We already apply FLIPBIT to memory that has frequent writes, so we expect this to be the usual case.

In Figures 17 and 18, we show the increase in flash lifetime for our benchmarks. The geometric mean lifetime increase is 68% and 44% for video and ML benchmarks, respectively. We notice that the trends of lifetime increase are very similar to the trends of energy reduction. This is expected since the majority of our energy reduction benefit stems from the fact that we are reducing the number of erases required by the device. At the same time, this erase reduction also increases our lifetime by reducing the number of program/erase cycles.

D. *Hardware Overhead*

We synthesized the n-bit approximation and error tracking hardware described in Section III-B in Synopsys Design

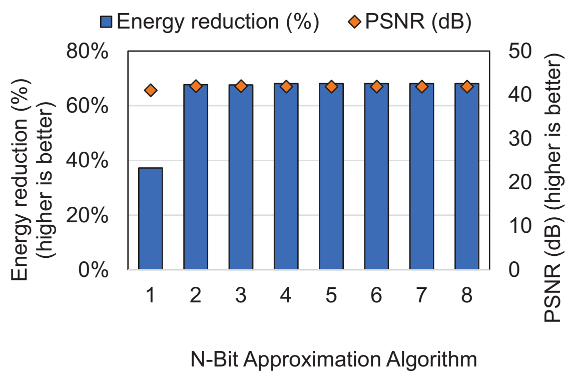


Fig. 16: The geometric mean of energy and PSNR for all videos while sweeping N in the N-bit approximation.

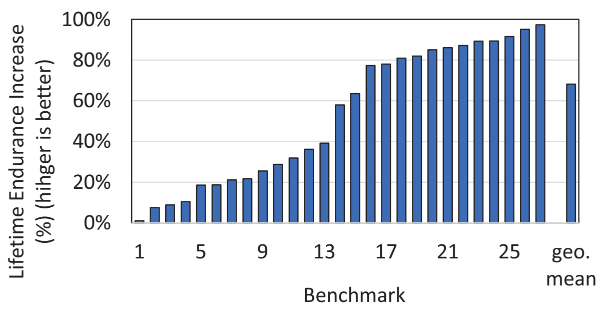


Fig. 17: Lifetime increase using 2-bit approximation for video benchmarks

Compiler 2017.09 in a 65 nm technology. We can synthesize the hardware up to 1 GHz, but since the flash runs at 33 MHz [75], Table IV shows the area and power when the circuit is constrained to a 33 MHz clock. We show two versions, the first is configurable for n from 1 to 8, while the second is hardcoded to n = 2. The latter version has reduced area and power because of logic optimizations not possible when ensuring configurability.

The circuit computes the output for one 8, 16, or 32-bit value, so for a 256 byte page, we would need to duplicate this circuit up to 256 times. However, due to the high latency caused by flash writes, we instead reuse the same hardware multiple times to save area. Since the flash runs at 33 MHz [75] and our hardware runs at up to 1 GHz, we perform all approximations while values are written to the buffer. Therefore, we do not impact the critical path of flash writes.

Compared to an ARM Cortex-M0+ SoC in the same 65 nm technology, our hardware uses only 0.1% area [64]. In energy, running the approximation for the *entire* page consumes up to 574 pJ, equal to 0.1% of the energy required to program a *single byte*. Given the low overheads of our proposed hardware, we believe it beneficial to add to IoT devices requiring low energy consumption.

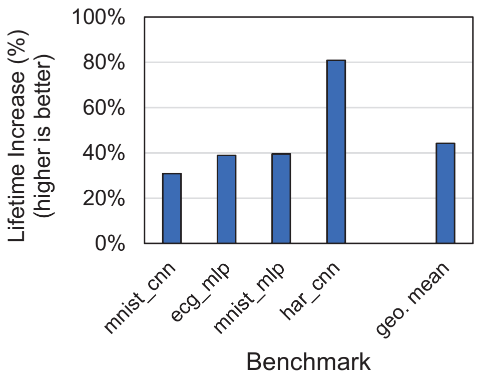


Fig. 18: Lifetime increase using 2-bit approximation for ML benchmarks.

TABLE IV: Hardware overhead for the N-bit approximation and error tracking hardware at 33 MHz.

N-Bit Approximation	**Area (μm^2)**	**Percentage of SoC [64]**	**Power (μW) at 33 MHz**
1–8 (configurable)	3919	0.104%	74.05
2	3213	0.0855%	69.20

VI. Broader Applicability of FlipBit

We now discuss other areas where FlipBit can be applied to benefit flash-based systems.

Floating-Point: We focus on integer and fixed point data types as they are more commonly used in low-power devices [29], [40], [71], [99]. However, FlipBit is easily extended to support floating-point numbers. For example, we can approximate the lower M bits of the mantissa, while keeping the sign and exponent bits precise. M is application dependent; more error tolerant applications have higher values of M. We would also need to replace the error calculation hardware (Figure 9) to use floating-point adders and subtractors. While, this increases the area, the overhead of FlipBit remains small as devices which support floating-point operations are typically larger.

Energy Harvesting: IoT devices are typically *persistently* powered (i.e., using wired power or batteries). However, energy-harvesting (EH) devices run directly on ambient energy (e.g., RF [34], [60], movement [6], vibration [44]). EH devices store energy in capacitors for short bursts of computation, after which the device remains off until enough energy is accumulated to perform more computation. When processing cannot be completed in a single "on-period", data must be saved to non-volatile memory (NVM). While EH platforms with emerging technologies such as ReRAM [61], [70] and STT-RAM [77] have been proposed in academia, commercial EH solutions continue to use regular IoT platforms, where flash continues to dominate [31]. Also, prior work has shown that energy-harvesting applications are also amenable to approximation [27], [55], [63]. Thus, FlipBit can also be used to approximate data backups to reduce energy and extend device lifetime in energy-harvesting IoT systems.

FlipBit for MLC: While we focus our approximation algorithms and evaluation on SLC due to its use in IoT devices [3], [88], we show how our technique could be extended to MLC flash. In MLC, a fully erased cell maps to 11; as the charge increases, the logical mapping decrements. Using program operations, we can only go from $11 \rightarrow 10 \rightarrow 01 \rightarrow 00$. To apply FlipBit to MLC, instead of making a decision one bit at a time, we would have to make it two bits at a time. This is since a 10_2 value could now become 01_2 through only a program operation. Specifically, we redefine our algorithms from n-*bit* to n-*cell* approximation algorithms. Our new n-cell approximation algorithm still does not look at more bits than it can set at each iteration and so functions similarly to the n-bit algorithm. We show how the 1-cell approximation algorithm would function using the same example as in Section III-A2. We are given *previous* = 0101 and *exact* = 0011. We first examine *previous*[3:2] = 01 and *exact*[3:2] = 00, we set *approx*[3:2] to 00. Now we examine *previous*[1:0] = 01 and *exact*[1:0] = 11. Since *exact*[1:0] $<$ *previous*[1:0], we set *approx*[1:0] to the *previous* value of 01. Our final result is *approx* = 0001.

VII. Related Work

Prior work looks at reducing erase operations for energy reduction and lifetime increase in flash through different strategies. Fazackerley *et al.* [25] propose masked overwriting for NOR flash to reduce the number of erase operations required. However unlike our work, they still require an erase to occur once each byte in a page has been written to once. Bittman *et al.* modify software data structures to reduce bit flips in phase change memory [10]. Their work could be extended to flash, but unlike FlipBit does not consider the asymmetry that a $1 \rightarrow 0$ is comparatively cheap in flash. Nonetheless, their approach is orthogonal to ours and could be combined to reduce bit flips further if the application of interest uses the data structures targeted by them. MicroVault [4] reduces bit flips by employing gray coding, but their technique only works for storing counter values.

Different coding has also been proposed to increase the lifetime of flash [39], [57], [58], [98]. These codes allow more writes to occur to the same region without requiring an erase, but they increase the memory footprint. RLD coding [57] also considers the inherent noise that exists from analog sensor values to produce codes, but the authors do not explore trading accuracy for energy savings. In our constrained IoT environment, an increased memory footprint is undesirable. Orthogonal to FlipBit, compression [45], [65], including approximate versions [72], has been explored to reduce the total memory traffic, and therefore number of erases needed. Caching for wear levelling also exists [13] but assumes large amounts of SRAM available that does not exist in IoT devices.

Several techniques for approximate storage have previously been developed. Some techniques [20], [80], [82], [89] use less reliable writing mechanisms such as lowering the voltage or reducing the number of programming pulses used. Others use faulty or worn out cells for approximate data [33], [46], [82]. However, none of these leverage the asymmetric program versus erase opportunity that flash offers.

In SSDs, the FTL is used to reduce wear out by employing wear leveling [19]. However, usually a separate MCU runs on the SSD solely to run the FTL, which does not work well for energy constrained IoT devices. Various file systems have been proposed to reduce the number of erases required [24], [26], [43], [94]. File systems can have large RAM footprints and run on an OS, which is often constraining to an IoT device. Additionally, while our technique is aimed at the lower power IoT domain, it does not preclude the use of these software approaches to reduce page erases.

VIII. Conclusion

Flash memory consumes significant energy performing erases required for a $0 \rightarrow 1$ bit transition. Erases additionally result in the eventual wear out of flash. We propose FlipBit; hardware for reducing page erases using approximation. FlipBit exploits the idea that we can prevent an erase by avoiding flipping a bit from $0 \rightarrow 1$. We are the first to use approximation to write values to flash based on the previous value such that we use only $1 \rightarrow 0$ transitions. We show that applying FlipBit in an IoT domain can reduce energy by 39% in ML applications. Additionally, our technique increases the flash lifetime by 68% for video streaming applications.

Acknowledgements

We thank the anonymous reviewers for their thoughtful and detailed reviews of this work. Thanks to member of the NEJ group for their valuable feedback on this paper. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2020-04179 and the NSERC COHESA Strategic Research Network. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program.

References

[1] N. N. Alajlan and D. M. Ibrahim, "TinyML: Enabling of inference deep learning models on ultra-low-power IoT edge devices for AI applications," *Micromachines*, vol. 13, no. 6, 2022.
[2] A. Alkhayyat, A. A. Thabit, F. A. Al-Mayali, and Q. H. Abbasi, "WBSN in IoT health-based application: Toward delay and energy consumption minimization," *Journal of Sensors*, vol. 2019, 2019.
[3] A. I. Alsalibi, M. K. Y. Shambour, M. A. Abu-Hashem, M. Shehab, Q. Shambour, and R. Muqat, *Nonvolatile Memory-Based Internet of Things: A Survey*. Cham: Springer International Publishing, 2022, pp. 285–304.
[4] E. Aras, M. Ammar, F. Yang, W. Joosen, and D. Hughes, "MicroVault: Reliable storage unit for IoT devices," in *16th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, 2020, pp. 132–140.
[5] ARM, "Cortex-M0+." [Online]. Available: https://developer.arm.com/Processors/Cortex-M0-Plus
[6] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini, "Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 1968–1980, 2016.
[7] S. Basu, L. Duch, M. Peón-Quirós, D. Atienza, G. Ansaloni, and L. Pozzi, "Heterogeneous and inexact: Maximizing power efficiency of edge computing sensors for health monitoring applications," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[8] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, 2003.

[9] A. Bhattacharyya, A. Somashekhar, and J. San Miguel, "NvMR: Non-volatile memory renaming for intermittent computing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. New York, NY, USA: Association for Computing Machinery, 2022, p. 1–13.

[10] D. Bittman, D. D. E. Long, P. Alvaro, and E. L. Miller, "Optimizing systems for Byte-Addressable NVM by reducing bit flipping," in *17th USENIX Conference on File and Storage Technologies (FAST 19)*. Boston, MA: USENIX Association, Feb. 2019, pp. 17–30.

[11] S. Boboila and P. Desnoyers, "Write endurance in flash drives: Measurements and analysis," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies*, ser. FAST'10. USA: USENIX Association, 2010, p. 9.

[12] D. Bortolotti, H. Mamaghanian, A. Bartolini, M. Ashouei, J. Stuijt, D. Atienza, P. Vandergheynst, and L. Benini, "Approximate compressed sensing: Ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 45–50.

[13] J. Boukhobza, P. Olivier, and S. Rubini, "A cache management strategy to replace wear leveling techniques for embedded flash memory," in *International Symposium on Performance Evaluation of Computer & Telecommunication Systems*, 2011, pp. 1–8.

[14] L. V. Cargnini, L. Torres, R. M. Brum, S. Senni, and G. Sassatelli, "Embedded memory hierarchy exploration based on magnetic random access memory," *Journal of Low Power Electronics and Applications*, vol. 4, no. 3, pp. 214–230, 2014.

[15] Y.-H. Chang, J.-H. Lin, J.-W. Hsieh, and T.-W. Kuo, "A strategy to emulate NOR flash with NAND flash," *ACM Trans. Storage*, vol. 6, no. 2, 7 2010.

[16] N. Chervyakov, P. Lyakhov, and N. Nagornov, "Analysis of the quantization noise in discrete wavelet transform filters for 3D medical imaging," *Applied Sciences*, vol. 10, no. 4, 2020.

[17] A. Chimenton and P. Olivo, "Reliability of erasing operation in NOR-flash memories," *Microelectronics Reliability*, vol. 45, no. 7, pp. 1094–1108, 2005.

[18] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Proceedings of the 47th Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2010, p. 555–560.

[19] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, "A survey of flash translation layer," *Journal of Systems Architecture*, vol. 55, no. 5, pp. 332–343, 2009.

[20] J. Cui, Y. Zhang, L. Shi, C. J. Xue, W. Wu, and J. Yang, "ApproxFTL: On the performance and lifetime improvement of 3-D NAND flash-based ssds," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 37, no. 10, p. 1957–1970, Oct 2018.

[21] H. Desai, M. Nardello, D. Brunelli, and B. Lucia, "Camaroptera: A long-range image sensor with local inference for remote sensing applications," *ACM Transactions on Embedded Computing Systems*, vol. 21, no. 3, 2022.

[22] Q. Dong, Y. Kim, I. Lee, M. Choi, Z. Li, J. Wang, K. Yang, Y.-P. Chen, J. Dong, M. Cho, G. Kim, W.-K. Chang, Y.-S. Chen, Y.-D. Chih, D. Blaauw, and D. Sylvester, "11.2 a 1Mb embedded NOR flash memory with 39µW program power for mm-scale high-temperature sensor nodes," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2017, pp. 198–199.

[23] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski, "Where's the bear? automating wildlife image processing using IoT and edge cloud systems," in *Proceedings of the Second International Conference on Internet-of-Things Design and Implementation*, ser. IoTDI '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 247–258.

[24] J. Engel and R. Mertens, "LogFS-finally a scalable flash file system," in *12th International Linux System Technology Conference*, 2005.

[25] S. Fazackerley, W. Penson, and R. Lawrence, "Write improvement strategies for serial NOR dataflash memory," in *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2016, pp. 1–6.

[26] E. Gal and S. Toledo, "A transactional flash file system for microcontrollers." in *USENIX Annual Technical Conference, General Track*, 2005, pp. 89–104.

[27] K. Ganesan, J. San Miguel, and N. Enright Jerger, "The What's Next intermittent computing architecture," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[28] D. Garcia and E. Barzilay. (2019) Crossover to memory expansion with Adesto ECOXiP and NXP's i.MX RT crossover processors. [Online]. Available: https://www.nxp.com/docs/en/white-paper/NXPADESTOWP.pdf

[29] S. Ge, Z. Luo, S. Zhao, X. Jin, and X.-Y. Zhang, "Compressing deep neural networks for efficient visual inference," in *IEEE International Conference on Multimedia and Expo (ICME)*, 2017, pp. 667–672.

[30] S. Gollakota, M. S. Reynolds, J. R. Smith, and D. J. Wetherall, "The emergence of RF-powered computing," *Computer*, vol. 47, no. 1, pp. 32–39, 2014.

[31] Grand View Research. (2022) Embedded non-volatile memory market size. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/embedded-non-volatile-memory-envm-market

[32] S. Ha and S. Choi, "Convolutional neural networks for human activity recognition using multiple accelerometer and gyroscope sensors," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016, pp. 381–388.

[33] L. Han, H. Amrouch, Z. Shao, and J. Henkel, "Rebirth-FTL: Lifetime optimization via approximate storage for NAND flash," in *IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2019, pp. 1–6.

[34] J. Hester, L. Sitanayah, and J. Sorber, "Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, 2015.

[35] M. Hicks, "Thumbulator: Cycle-accurate ARMv6-M simulator," 2018. [Online]. Available: https://github.com/impedimentToProgress/thumbulator

[36] D. Ielmini, "Reliability issues and modeling of flash and post-flash memory (invited paper)," *Microelectronic Engineering*, vol. 86, no. 7, pp. 1870–1875, 2009, iNFOS 2009.

[37] Infineon. (2021) Programmer's guide for the hyperflash family. [Online]. Available: https://www.infineon.com/dgdl/Infineon-AN99195_Programmer_s_Guide_for_the_HyperFlash_Family-ApplicationNotes-v05_00-EN.pdf?fileId=8ac78c8c7cdc391c017d074116b66430

[38] Infineon. (2022) 512 Mb (64 MB), 3.0 V Serial NOR Flash Memory. [Online]. Available: https://www.infineon.com/dgdl/Infineon-CYWT16B512-133FZQB-DataSheet-v01_00-EN.pdf?fileId=8ac78c8c8412f8d3018456c4f4ae09c1

[39] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Writing cosets of a convolutional code to increase the lifetime of flash memory," in *50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2012, pp. 308–318.

[40] P. Judd, J. Albericio, T. Hetherington, T. M. Aamodt, N. Enright Jerger, and A. Moshovos, "Proteus: Exploiting numerical precision variability in deep neural networks," in *Proceedings of the International Conference on Supercomputing*. New York, NY, USA: Association for Computing Machinery, 2016.

[41] J. Kim, J. Lee, S. Lee, and M. Lee, "Development of 3-D stereo endoscopic PACS viewer," in *IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, vol. 1. IEEE, 2001, pp. 278–280.

[42] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[43] C. Lee, D. Sim, J. Hwang, and S. Cho, "F2FS: A new file system for flash storage," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, Feb. 2015, pp. 273–286.

[44] S. Lee, B. Islam, Y. Luo, and S. Nirjon, "Intermittent learning: On-device machine learning on intermittently powered system," *Proceedings of the ACM conference on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2019.

[45] S. Lee, J. Park, K. Fleming, Arvind, and J. Kim, "Improving performance and lifetime of solid-state drives using hardware-accelerated compression," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 4, pp. 1732–1739, 2011.

[46] F. Li, Y. Lu, Z. Wu, and J. Shu, "ASCache: An approximate SSD cache for error-tolerant applications," in *Proceedings of the 56th Annual Design Automation Conference*. New York, NY, USA: Association for Computing Machinery, 2019.

[47] Q. Li, Z. Liu, and J. Xiao, "A data collection collar for vital signs of cows on the grassland based on LoRa," in *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*, 2018, pp. 213–217.

[48] X. Li, R. Lu, X. Liang, X. Shen, J. Chen, and X. Lin, "Smart community: an internet of things application," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 68–75, 2011.

[49] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: Memory-efficient patch-based inference for tiny deep learning," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[50] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 2999–3007.

[51] Z. Liqiang, Y. Shouyi, L. Leibo, Z. Zhen, and W. Shaojun, "A crop monitoring system based on wireless sensor network," *Procedia Environmental Sciences*, vol. 11, pp. 558–565, 2011.

[52] Y. Liu, Z. Wang, A. Lee, F. Su, C.-P. Lo, Z. Yuan, C.-C. Lin, Q. Wei, Y. Wang, Y.-C. King, C.-J. Lin, P. Khalili, K.-L. Wang, M.-F. Chang, and H. Yang, "4.7 A 65nm ReRAM-enabled nonvolatile processor with 6× reduction in restore time and 4× higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2016, pp. 84–86.

[53] H.-T. Lue, T.-H. Hsu, T.-H. Yeh, W.-C. Chen, C. R. Lo, C.-T. Huang, G.-R. Lee, C.-J. Chiu, K.-C. Wang, and C.-Y. Lu, "A Vertical 2T NOR (V2T) Architecture to Enable Scaling and Low-Power Solutions for NOR Flash Technology," in *IEEE Symposium on VLSI Technology*, 2020, pp. 1–2.

[54] P. Lussier and C.-H. Yu, "Applying IoT and deep learning for ECG data analysis," in *IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2022, pp. 01–06.

[55] K. Ma, X. Li, J. Li, Y. Liu, Y. Xie, J. Sampson, M. T. Kandemir, and V. Narayanan, "Incidental computing on IoT nonvolatile processors," in *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[56] K. Ma, Y. Zheng, S. Li, K. Swaminathan, X. Li, Y. Liu, J. Sampson, Y. Xie, and V. Narayanan, "Architecture exploration for ambient energy harvesting nonvolatile processors," in *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 526–537.

[57] G. Mappouras, A. Vahid, R. Calderbank, and D. J. Sorin, "Extending flash lifetime in embedded processors by expanding analog choice," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2462–2473, 2018.

[58] F. Margaglia, G. Yadgar, E. Yaakobi, Y. Li, A. Schuster, and A. Brinkmann, "The devil is in the details: Implementing flash page reuse with WOM codes," in *14th USENIX Conference on File and Storage Technologies (FAST 16)*. Santa Clara, CA: USENIX Association, Feb. 2016, pp. 95–109.

[59] G. V. Merrett and B. M. Al-Hashimi, "Energy-driven computing: Rethinking the design of energy harvesting systems," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017, pp. 960–965.

[60] A. Mirhoseini, E. M. Songhori, and F. Koushanfar, "Idetic: A high-level synthesis approach for enabling long computations on transiently-powered ASICs," in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 2013.

[61] C. S. Mishra, J. Sampson, M. T. Kandemir, and V. Narayanan, "Origin: Enabling on-device intelligence for human activity recognition using energy harvesting wireless sensor networks," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2021.

[62] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, mar 2016.

[63] A. Montanari, M. Alloulah, and F. Kawsar, "Degradable inference for energy autonomous vision applications," in *Adjunct Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the ACM International Symposium on Wearable Computers*, 2019.

[64] J. Myers, A. Savanth, D. Howard, R. Gaddh, P. Prabhat, and D. Flynn, "An 80nW retention 11.7pJ/cycle active subthreshold ARM Cortex-M0+ subsystem in 65nm CMOS for WSN applications," in *IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*, 2015, pp. 1–3.

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

[66] M. O. Ojo, S. Giordano, G. Procissi, and I. N. Seitanidis, "A review of low-end, middle-end, and high-end Iot devices," *IEEE Access*, vol. 6, pp. 70 528–70 554, 2018.

[67] M. Olafenwa, "ImageAI, an open source python library built to empower developers to build applications and systems with self-contained computer vision capabilities," 2018. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI

[68] Y. Pan, Z. Hu, N. Zhang, H. Hu, W. Xia, Z. Jiang, L. Shi, and S. Li, "HNFFS: Revisiting the NOR flash file system," in *IEEE 11th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, 2022, pp. 14–19.

[69] S. Payvar, M. Khan, R. Stahl, D. Mueller-Gritschneder, and J. Boutellier, "Neural network-based vehicle image classification for IoT devices," in *IEEE International Workshop on Signal Processing Systems (SiPS)*, 2019, pp. 148–153.

[70] K. Qiu, N. Jao, M. Zhao, C. S. Mishra, G. Gudukbay, S. Jose, J. Sampson, M. T. Kandemir, and V. Narayanan, "ResiRCA: A resilient energy harvesting ReRAM crossbar-based accelerator for intelligent embedded processors," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[71] V. Rajagopal, C. K. Ramasamy, A. Vishnoi, R. N. Gadde, N. R. Miniskar, and S. K. Pasupuleti, "Accurate and efficient fixed point inference for deep neural networks," in *25th IEEE International Conference on Image Processing (ICIP)*, 2018, pp. 1847–1851.

[72] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate memory compression," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 980–991, 2020.

[73] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv*, 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1804.02767

[74] Renesas. (2019) NOR flash memory erase operation. [Online]. Available: https://www.renesas.com/us/en/document/apn/an500nor-flash-memory-erase-operation

[75] Renesas. (2022) AT25XE161D: 16-Mbit 1.65 V – 3.6 V Range SPI Serial Flash Memory with Multi I/O Support. [Online]. Available: https://www.renesas.com/us/en/document/dst/at25xe161d-datasheet?language=en

[76] Renesas. (2023) AT45DB161E 16-Mbit DataFlash (with Extra 512-kbits) 2.3 V or 2.5V Minimum SPI Serial Flash Memory. [Online]. Available: https://www.renesas.com/in/en/document/dst/at45db161e-datasheet?r=1608911

[77] S. Resch, S. K. Khatamifard, Z. I. Chowdhury, M. Zabihi, Z. Zhao, H. Cilasun, J.-P. Wang, S. S. Sapatnekar, and U. R. Karpuzcu, "Mouse: Inference in non-volatile memory for energy harvesting applications," in *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020.

[78] P. Roy, R. Ray, C. Wang, and W. F. Wong, "ASAC: Automatic sensitivity analysis for approximate computing," in *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*. New York, NY, USA: Association for Computing Machinery, 2014, p. 95–104.

[79] E. Ruppel, M. Surbatovich, H. Desai, K. Maeng, and B. Lucia, "An architectural charge management interface for energy-harvesting systems," in *55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 318–335.

[80] M. Salajegheh, Y. Wang, K. Fu, A. A. Jiang, and E. Learned-Miller, "Exploiting Half-Wits: Smarter storage for Low-Power devices," in *9th USENIX Conference on File and Storage Technologies (FAST 11)*. San Jose, CA: USENIX Association, Feb. 2011.

[81] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY, USA: Association for Computing Machinery, 2011, p. 164–174.

[82] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM Trans. Comput. Syst.*, vol. 32, no. 3, sep 2014.

[83] J. Santos, T. Wauters, B. Volckaert, and F. De Turck, "Resource provisioning for IoT application services in smart cities," in *13th International Conference on Network and Service Management (CNSM)*, 2017, pp. 1–9.

[84] S. Semiconductor. (2017) ARTIK 053/053s module datasheet. [Online]. Available: https://www.mouser.com/catalog/specsheets/Samsung_11292017_SIP-0P5WRS302.pdf

[85] M. Shoushtari, A. BanaiyanMofrad, and N. Dutt, "Exploiting partially-forgetful memories for approximate computing," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 19–22, 2015.

[86] STMicroelectronics. (2017) Access line ultra-low-power 32-bit MCU Arm®-based Cortex®-M0+, up to 16KB Flash, 2KB SRAM, 512B EEPROM, ADC. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l011k4.pdf

[87] STMicroelectronics. (2022) RM0091 reference manual. [Online]. Available: https://www.st.com/resource/en/reference_manual/rm0091-stm32f0x1stm32f0x2stm32f0x8-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

[88] Super Talent. SLC vs. MLC: An analysis of flash memory. [Online]. Available: http://www.supertalent.com/datasheets/SLC_vs_MLC%20whitepaper.pdf

[89] M. T. Teimoori, M. A. Hanif, A. Ejlali, and M. Shafique, "AdAM: Adaptive approximation management for the non-volatile memory hierarchies," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 785–790.

[90] B. Thoen, G. Callebaut, G. Leenders, and S. Wielandt, "A deployable LPWAN platform for low-cost and energy-constrained IoT applications," *Sensors*, vol. 19, no. 3, 2019.

[91] S. Trilles, A. González-Pérez, and J. Huerta, "An IoT platform based on microservices and serverless paradigms for smart farming purposes," *Sensors*, vol. 20, no. 8, 2020.

[92] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, 2014, pp. 27–32.

[93] M. Venzke, D. Klisch, P. Kubik, A. Ali, J. D. Missier, and V. Turau, "Artificial neural networks for sensor data classification on small embedded systems," *arXiv*, 2020. [Online]. Available: https://doi.org/10.48550/arXiv.2012.08403

[94] D. Woodhouse, "JFFS: The journalling flash file system," in *Ottawa linux symposium*, vol. 2001, 2001.

[95] M. Xie, C. Pan, Y. Zhang, J. Hu, Y. Liu, and C. J. Xue, "A novel STT-RAM-based hybrid cache for intermittently powered processors in IoT devices," *IEEE Micro*, vol. 39, no. 1, pp. 24–32, 2019.

[96] Xiph.org, "Video test media." [Online]. Available: https://media.xiph.org/video/derf/

[97] M. Xu, X. Zhang, Y. Liu, G. Huang, X. Liu, and F. X. Lin, "Approximate query service on autonomous IoT cameras," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. New York, NY, USA: Association for Computing Machinery, 2020, p. 191–205.

[98] G. Yadgar, E. Yaakobi, and A. Schuster, "Write once, get 50% free: Saving SSD erase costs using WOM codes," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*. Santa Clara, CA: USENIX Association, Feb. 2015, pp. 257–271.

[99] A. H. Zadeh, M. Mahmoud, A. Abdelhadi, and A. Moshovos, "Mokey: Enabling narrow fixed-point inference for out-of-the-box floating-point transformer models," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*. New York, NY, USA: Association for Computing Machinery, 2022, p. 888–901.

[100] H. Zhang, M. Putic, and J. Lach, "Low power GPGPU computation with imprecise hardware," in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6.

[101] I. A. Zualkernan, S. Dhou, J. Judas, A. R. Sajun, B. R. Gomez, L. A. Hussain, and D. Sakhnini, "Towards an IoT-based deep learning architecture for camera trap image classification," in *IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2020, pp. 1–6.