Today’s intermittent computing systems operate by relying only on harvested energy accumulated in their tiny energy reservoirs, typically capacitors. An intermittent device dies due to a power failure when there is no energy in its capacitor and boots again when the harvested energy is sufficient to power its hardware components. Power failures prevent the forward progress of computation due to the frequent loss of computational state. To remedy this problem, intermittent computing systems comprise built-in fast non-volatile memories with high write endurance to store information that persists despite frequent power failures. However, the lack of design tools makes fast-prototyping these systems difficult. Even though FPGAs are common platforms for fast prototyping and behavioral verification of continuously-powered architectures, they do not target prototyping intermittent computing systems. This article introduces a new FPGA-based framework, named NORM (Non-volatile memORY eMulator), to emulate and verify the behavior of any intermittent computing system that exploits fast non-volatile memories. Our evaluation showed that NORM can be used to emulate and validate FeRAM-based transiently-powered hardware architectures successfully.

CCS Concepts:
- Hardware → Energy generation and storage; Reconfigurable logic applications;
- Computer systems organization → Embedded hardware; Special purpose systems.

Additional Key Words and Phrases: Intermittent computing, Non-volatile Processors, FPGAs.

ACM Reference Format:
Simone Ruffini, Luca Caronti, Kasım Sinan Yıldırım, and Davide Brunelli. 2022. NORM: An FPGA-based Non-volatile Memory Emulation Framework for Intermittent Computing. ACM J. Emerg. Technol. Comput. Syst., Vol. 1, No. 1, Article 1 (January 2022), 19 pages. https://doi.org/10.1145/3517812

1 INTRODUCTION
The recent advancements in microelectronics led to the emergence of batteryless sensors that operate relying only on ambient energy [40]. This sensing technology opens up new application spaces where small devices should have eternal lifetimes, autonomous operation, and massive deployments in inaccessible locations [14]. Batteryless sensors comprise energy harvesting circuits that use several sources such as solar, thermal, and radio waves to accumulate the environmental

Authors’ addresses: Simone Ruffini, University of Trento, Department of Information Engineering and Computer Science, Via Sommarive, 9, Povo, 38123 TN, Trento, Italy, simone.ruffini@studenti.unitn.it; Luca Caronti, University of Trento, Department of Information Engineering and Computer Science, Via Sommarive, 9, Povo, 38123 TN, Trento, Italy, luca.caronti@studenti.unitn.it; Kasım Sinan Yıldırım, University of Trento, Department of Information Engineering and Computer Science, Via Sommarive, 9, Povo, 38123 TN, Trento, Italy, kasimsinan.yildirim@unitn.it; Davide Brunelli, University of Trento, Department of Industrial Engineering, Via Sommarive, 9, Povo, 38123 TN, Trento, Italy, davide.brunelli@unitn.it.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2022 Association for Computing Machinery.
1550-4852/2022/1-ART1 $15.00
https://doi.org/10.1145/3517812
energy into a small energy buffer, typically a tiny capacitor. When the stored harvested energy is above an operating threshold, the microcontroller reboots to compute, sense, and communicate. When the energy drains out of the capacitor, the microcontroller and peripherals turn off due to a power failure. Today’s batteryless sensors are composed of ultra-low-power microcontrollers whose main architectural components are volatile. When the batteryless sensor turns off due to a power failure, the volatile processor state (e.g., the contents of the stack, program counter, registers) is lost. This operation leads to the loss of all computational states and intermediate results [38].

The software on batteryless platforms runs intermittently due to frequent charge-discharge cycles. As a consequence of the intermittent execution, the computation might not progress forward and memory consistency might be violated [37]. To store information that will persist despite power failures, microcontrollers in batteryless sensors comprise embedded non-volatile secondary memory components, e.g., FeRAM [7, 8] that exhibits low-power characteristics, faster write performance and greater maximum read/write endurance compared to Flash memories, even if they pay lower memory density than other recent NVM technologies. Using software-based techniques (e.g., [2, 18, 62]), programmers backup the volatile state of the microcontroller into non-volatile memory to recover computation from where it left upon a power failure. As an alternative to software-based solutions, leveraging non-volatile logic and building non-volatile processors (NVPs) is another approach to ensure forward progress of computation and keep memory consistent during intermittent execution. NVPs integrate built-in non-volatile memory in their architecture. They automatically back up the computation state into their internal non-volatile registers upon a power failure and restore the state upon recovery [24]. All these operations are transparent to the programmer.

The architectural design space of intermittent computing systems that exploit non-volatile logic is broad and includes several design options with different pros and cons. As an example, a crucial design decision is to identify which state elements will be non-volatile. Systems designers can keep all registers non-volatile, which is slower and more energy-consuming. Alternatively, the designers can keep all registers as volatile, but they can maintain additional non-volatile registers to back up the volatile state (i.e., volatile registers) at specific points in time. Another crucial issue is to decide the backup frequency of the volatile state components. For instance, a computing system can backup its state at every clock cycle, or it can backup on-demand [24, 25], to decrease the backup frequency and save energy. However, the lack of design tools makes fast-prototyping and functional verification of computing systems with non-volatile logic difficult. FPGAs (field-programmable gate arrays) are useful for fast prototyping and verification of digital logic. As of now, FPGA fabrics include logic elements that are implemented using either volatile memory or non-volatile memory [47], but not both. Therefore, existing HDLs (such as VHDL or Verilog) do not provide specific keywords to make a differentiation between a volatile state element and a non-volatile state element. This situation prevents hardware designers from using FPGAs to fast-prototype their logic designs targeting intermittent computing, which include both volatile and non-volatile logic. To the best of our knowledge, the state-of-the-art does not propose a solution to emulate transiently-powered intermittently operating hardware architectures using off-the-shelf FPGAs.

In this article, we introduce a new FPGA-based framework, named NORM (Non-volatile memORy eMulator), that can be used to emulate any intermittent computing system with fast non-volatile memory. NORM can be used to debug and perform functional verification of non-volatile computing logic. Moreover, NORM can be integrated into a working intermittent computing system in place of a yet-to-be-built non-volatile computing logic so the whole system can be tested. NORM comprises auxiliary blocks that:
Fig. 1. This figure presents the operation of an energy harvesting battery-less computing device. The device harvests energy and stores it in a capacitor. The device dies when the voltage level of the capacitor is below a threshold. The device boots again when the voltage level is above this threshold. This operation leads to intermittent computation. The volatile micro-controller is equipped with external non-volatile memory to store intermediate results and computation state to recover from power failures.

(1) simulates the behavior of irregular power supply typical to energy-harvesting intermittent systems;
(2) simulates the persistence of the non-volatile micro-architectural elements as well as the long delay of reading/write operations (as compared to those of volatile memory);
(3) approximates the power consumption of the emulated technology.

Our simulations showed that NORM can be used to emulate and validate FeRAM-based transiently-powered hardware architectures successfully. We release the source code of NORM (implemented in VHDL) in a public repository [32] to increase the impact of this work and enable the community to fast prototype and validate transiently-powered non-volatile hardware architectures.

The rest of this article is organized as follows. In Section 2, we provide the related work on intermittent computing and NVPs. We present the general description of NORM in Section 3. Section 4 presents the implementation details of NORM and Section 5 presents our evaluation based on simulations. Finally, Section 6 concludes our article and proposes future work.

2 BACKGROUND AND RELATED WORK

A new class of embedded devices that can sense, compute, and communicate without batteries emerged. As an example, RF-powered batteryless sensors [11, 51] solely rely on the harvested energy of ambient radio frequency waves in the air (see Figure 1). These batteryless devices, which can feature even more complex sensors such as cameras [31], comprise ultra-low-power microcontrollers (e.g., MSP430FR5969 [49]) whose main architectural components, e.g., registers and main memory, are volatile. These volatile processors include also a non-volatile secondary memory, e.g., Ferroelectric RAM (FRAM) [50], to store information that will persist upon power failures. Despite several ultra-low-power operation modes of these microcontrollers (e.g., sleep mode requires current on the order of a few μA), batteryless sensors cannot be available continuously using unreliable and sporadic energy sources [1, 41]. Frequent and unpredictable power failures reset the volatile state of the device, prevent the forward progress of computation and hinder its memory consistency. Therefore, programs and libraries designed for continuously-powered computers cannot run on batteryless sensors correctly due to the frequent loss of volatile state, and in turn, failed computation.
2.1 Intermittent Computing with Volatile Processors

Volatile microcontrollers employ software-aided solutions to mitigate the effects of unpredictable power failures. Generally speaking, these solutions backup the volatile state of the processor into non-volatile memory to recover computation from where it left upon a power failure. Moreover, they ensure memory consistency so that the backed-up state in the non-volatile memory will not be different than the volatile state, or vice versa. Current literature proposed mainly two software-based approaches. One approach is to store the computation state in non-volatile memory via checkpoints paired with C programs [2, 3, 16–18, 23, 27, 38, 52]. Upon recovery from a power failure, the computation continues from the consistent volatile state stored in the latest successful checkpoint. Another approach is to use a custom task-based programming model to develop intermittent applications, which eliminates the high cost of checkpointing [4, 15, 26, 28, 29, 39, 62]. In this model, programmers implement the applications as a collection of idempotent and atomic tasks by employing an explicit task-based control flow. Individual task sizes should not exceed the capacity of the capacitor to ensure forward progress. However, software-aided recovery solutions require transmitting data from built-in volatile components of the processor, e.g., registers, to non-volatile memory. This operation suffers from low speed, e.g., 200 𝜇s [42], and a large energy penalty that grows with the size of volatile elements [42, 45]. Moreover, these solutions require programmers to structure their software by considering programming models designed for intermittent systems, e.g., task-based programming [4, 62].

2.2 Non-volatile Logic and Processors

Non-volatile processors (NVPs) bring non-volatile memory into the micro-architecture of the processor. Non-volatile logic enables the backup and recovery operations from a power failure to be transparent to the programmer. Moreover, backup and recovery introduce less overhead than software-aided solutions, e.g., only on the order of a few 𝜇s [20, 57]. Since backup and retention operations are fast as compared to the software-aided solutions, NVPs reduce leakage power by shutting down the system when the device idle [1].

Due to the higher power required for non-volatile memory read/write operations, NVPs might also consume more power as compared to volatile processors [25]. Therefore, there is room for micro-architecture-level optimizations to reduce their energy consumption. To decrease the energy requirements of NVPs, recent works proposed:

1. using more efficient memory technologies, e.g., ReRAM [22] and hybrid CMOS/ferroelectric non-volatile flipflop [44];
2. embedding non-volatility into the computing logic, e.g., transistor level, using NCFET [20] so that logic gates could also store their states intrinsically in a non-volatile fashion;
3. using new backup strategies, e.g., backup at every processor cycle or on-demand backup [24, 25], to decrease backup frequency to save energy.

These efforts provide implementation technology-level energy optimizations. However, as of now, we do not have tools to fast prototype non-volatile logic and observe optimization strategies targeting different non-volatile intermittent computing architectures and processors.

2.3 Non-volatile Memory Simulation/Emulation Frameworks

There are studies, e.g., [9, 19, 34, 63], that provide the emulation of different non-volatile main memory technologies. These studies present techniques to assess the system’s performance concerning different non-volatile memory technologies. Unfortunately, they do not apply to intermittent computing systems. There are studies, e.g., [12, 36], that proposed simulators for non-volatile memory and logic. As an example, NVPSim [12] can simulate the architectural components forming
a non-volatile processor by allowing users to select only different configurations (e.g., cache size and organization) for the main high-level components in a processor. Fused [43] is designed to assess the performance of intermittent systems by providing simulations only at a high level of abstraction. Authors in [58] describe a system-level simulator supporting flexible energy behavior configuration for both the processor and peripherals.

Contrarily to the mentioned studies, this work enables the assessment of non-volatile features in any digital hardware design that includes a combination of volatile and non-volatile logic. We provide full flexibility for the users to evaluate, validate and fast-prototype any HDL design targeting intermittent computing systems.

2.4 Field-programmable Gate Arrays (FPGAs)

FPGAs are used in many applications due to the increased cost and time associated with the custom ASIC (application-specific integrated circuit) design. FPGAs are useful for fast prototyping custom processor architectures and their behavioral verification. Several popular volatile processor architectures, such as RISC-V, have implementations using popular hardware description languages (HDLs) (e.g., Verilog) that can run on FPGAs. OpenFPGA framework [46] opened the door for automating the design, verification, and layout of different FPGA architectures. OpenFPGA enabled end-users to port their designs to any FPGAs that OpenFPGA can support. Some recent studies target reducing the energy consumption of FPGAs. As an example, the authors in [47] proposed an RRAM-based FPGA architecture, which is inherently fully non-volatile. They replaced the SRAM-based circuits in FPGA architectures with RRAM-based implementations. RRAM-based FPGAs can be powered off during sleep mode and instantly powered on when needed. This strategy reduces the energy requirements.

Using FPGAs to prototype intermittent computing architectures is an open issue. Hardware designs that operate intermittently are composed of volatile and non-volatile logic elements. Current FPGAs provide either volatile or non-volatile state elements, but not both. We do not have mixed memory volatility in the fabric of FPGA architectures. Hence, there are no specific HDL keywords to differentiate a non-volatile register from a volatile one. As of now, hardware designers cannot represent hardware that operates intermittently by using existing HDLs. They cannot validate their designs through simulations. This work focuses on these deficiencies and fills the existing gap in the literature by proposing a novel framework that facilitates the design and validation of intermittently-operating hardware.

3 NORM SYSTEM OVERVIEW

We propose an emulation architecture, named NORM (Non-volatile memORy eMulation), that mimics non-volatile memory elements and power failures. NORM architecture is composed of three main auxiliary blocks:

1. **Intermittency Emulation** that emulates irregular power supply typical to energy-harvesting systems;
2. **Non-volatile Register Emulation** that emulates the persistence of the non-volatile registers in the micro-architecture, and the delays of the read/write operations;
3. **Energy Consumption Approximation** that approximates the energy consumption of the emulated technology.

Figure 2 presents an overview of the proposed architecture. In the following subsections, we summarize the design of the aforementioned auxiliary blocks.
Fig. 2. Non-volatile Logic Emulation. Our architecture is composed of three auxiliary blocks. **Intermittency Emulation** emulates intermittent power supply. **Non-volatile Register Emulation** emulates the persistence of the non-volatile registers and the delays of the read/write operations. **Energy Approximation** approximates the energy consumption of the emulated technology.

Fig. 3. The computing system dies when the voltage level in the capacitor is below the reset threshold. The system starts operating when the voltage level is above the wake-up threshold. The software running on the computing system can also be notified, via the checkpoint threshold, so that the volatile state can be copied manually to the non-volatile memory, as in [2].

### 3.1 Intermittency Emulation

This block triggers a **Reset** signal to emulate a power failure. It can generate random triggers as well as follow an energy trace of a realistic energy harvesting scenario, as presented in [13]. This block can comprise a memory that can hold a pre-collected voltage trace. Thanks to a prescaler it is possible to choose the frequency at which a new voltage value can be read from this memory. The value can be compared against several pre-determined threshold values using comparators. If the voltage value in the trace is smaller than the corresponding threshold value, the corresponding comparator outputs a high signal. A multiplexer placed in front of the comparators selects the output of the desired threshold comparison operation as the **Reset** signal.
Table 1. Main parameters characterizing different non-volatile memory technologies.

| Feature / Technology      | FeRAM | MRAM  | nvSRAM | ReRAM | PRAM |
|---------------------------|-------|-------|--------|-------|------|
| Data retention (years)    | 10 [8]| 20 [33]| 20 [6]| 10 [10]| 10 [30]|
| Endurance (cycles)        | $10^{15}$ [8]| $10^8$ [48]| Unlimited [6]| $10^6$ [10]| $10^6$ [30]|
| Read access time (ns)     | 55 [7]| 35 [33]| 10 [5]| 10 [21]| 115 [30]|
| Write access time (ns)    | 55 [7]| 35 [33]| 10 [5]| 50 [21]| 115 [30]|
| Sizes (nm)                | 130 [53]| 14 [54]| 28 [55]| 90 [56]|
| Read Current (mA)         | 8 [7]| 55 [33]| 3 [6]| 1.5 [10]| 30 [30]|
| Write Current (mA)        | 8 [7]| 105 [33]| 3 [6]| 0.15 [10]| 15 [30]|
| Standby Current (μA)      | 90 [7]| 18000 [33]| 250 [6]| 60 [10]| 80 [30]|
| Sleep Current (μA)        | 5 [7]| 8 [6]| 6 [10]|
| Read energy 1 (pJ)        | 1452 | 6352.5| 99    | 49.5  | 11385|
| Write energy 2 (pJ)       | 1452 | 12127.5| 99    | 24.75 | 5692.5|

This block can also output signals that indicate if a threshold condition is satisfied to be used by some other logic in the architecture, as presented in Figure 3. In particular, a threshold value can be set to trigger a software routine just before a power failure, as in [2]. Preferentially, this block can signal a dedicated hardware module to trigger a backup operation. The signaled hardware block can copy the volatile state elements of the micro-architecture to their non-volatile counterparts.

3.2 Non-volatile Register Emulation.

Since the registers in FPGA logic elements are volatile, one cannot directly implement non-volatile registers. Within this block, we implement non-volatile registers using FPGAs’ volatile registers via the following design: In our emulation architecture, we connected the Reset signal to all volatile registers and combinational logic, i.e., to all components other than the non-volatile registers. Therefore, when the reset signal is triggered, volatile registers are cleared. Since we did not connect the Reset signal to non-volatile register blocks (implemented as volatile registers), their contents will remain in case of resetting other logic elements. Another issue is that non-volatile read/write operations are slower than volatile ones. We added logic that emulates the parametric delays introduced by non-volatile memory circuits. To this end, we placed a logic following the inputs of each volatile register, which emulates non-volatile memory delay.

3.3 Energy Consumption Approximation

We accelerate the energy estimation by mapping the energy model-related circuit onto the prototyping platform to provide a reliable emulation and assessment of the intermittent micro-architecture. We defined an energy model for each volatile logic block, as depicted in Figure 2. The energy model, fed by activity counters, is configured to measure the energy consumption of the programmed logic into each block. It considers the technology of the emulated chip and the type of activity requested by each volatile logic block. The counters provide in-depth and distributed information about the energy performance. The parametric delays introduced in the read/write NVMs realize the latency of the used memory technology (i.e., ReRAM, FRAM, etc.).

NORM implements the non-volatile memory energy consumption model by considering the real-world and already validated parameters provided by the vendors of the non-volatile memories. Table 1 presents a comparison of five different non-volatile memory technologies based on the main parameters characterizing them. We considered Ferroelectric RAM (FeRAM), Magnetoresistive RAM (MRAM), Non-Volatile SRAM (nvSRAM), Resistive RAM (ReRAM) and Phase-change RAM.
Fig. 4. Non-Volatile Register (NVR) block diagram. This entity groups: non-volatile memory emulator (NVRE), volatile register (BRAM), volatile memory cleaner (RESET BLOCK), and the multiplexer that imposes the right input signals to the volatile memory. RESET and POWER_RESET signals are the control inputs of the memory multiplexer and output multiplexer. Memory multiplexer selects signals for the input ports of the volatile register while the output multiplexer imposes either the volatile register output or zeros as the data output of NVR. The zero condition is met for both multiplexers only when POWER_RESET is on and RESET off.

(PRAM) technologies, by obtaining parameters from the state of the art commercial chips. The accuracy of NORM energy approximation (including the timing behavior) depends on the accuracy of these parameters that characterize the selected non-volatile memory technology.

4 NORM IMPLEMENTATION

After presenting the high-level description of NORM, we present its implementation details in this section. We implemented NORM in VHDL using Vivado 2020.1 [59], as a framework to emulate any digital non-volatile logic on FPGAs in the market. NORM framework helps designers of transiently-powered systems to test and characterize their architecture by using the provided non-volatile memory abstractions. The implementation of NORM framework is composed of a set of modules that implements the design presented in Section 3. These modules are:

1. Non-volatile Register (NVR),
2. Intermittency Emulator (IE),
3. Energy Approximator (EA)
4. Instant Energy Calculator (IEC).

In the following subsections, we describe the implementation of these modules in detail.

4.1 Non-Volatile Register (NVR)

The main blocks, input/output signals, and internal connections of the non-volatile register (NVR) are presented in Figure 4. We implemented the memory that holds NVR data using Xilinx block memory (BRAM) proprietary IP [61], but other open source BRAM implementations can also be
used. Inputs to a BRAM block are address bus (ADDR), input data (DIN), clock (CLK), write enable (WE) and enable (EN). These signals are also the general inputs of the NVR, as depicted in Figure 4. Apart from these signals, RESET is the internal hardware reset of the FPGA, and POWER_RESET is the reset signal that emulates the power failure (from intermittency emulator block, which we will present later). It is worth mentioning that POWER_RESET does not erase the volatile memory of the block. NVR also holds a non-volatile register emulator (NVRE) block and a reset block (RB).

4.1.1 Non-volatile Register Emulator (NVRE). Without the non-volatile register emulator (NVRE) block, NVR behaves like an ordinary volatile memory/register. NVRE imposes a strict access policy that takes into account non-volatile memory is slower than volatile one. This entity is instantiated by providing a time delay (expressed in nanoseconds), which defines the access delay of the emulated non-volatile memory. Hence, this component expects a scaled access time concerning system clock speed. The access time (i.e., the delays due to read and write operations) can easily be obtained from the data sheets of non-volatile memory components, as depicted in Table 1. It is also worth mentioning that the aging of the non-volatile memories is not a concern for our framework since the new memory technologies have a high write endurance. As an example, FeRAM has $10^{15}$ write endurance. Therefore, even 150000 write operations per second will lead to almost 211 years lifetime.

The delayed access time is enabled by a busy signal that informs endpoints about the operational status of the component. NVRE has three input signals, i.e., clock (CLK), reset (RST) and enable (EN). NVRE implements the following emulation protocol:

1. The input signal EN is used to enable NVR access. This signal is also connected to the main BRAM block that holds the NVR data. EN is captured on the rising edge of the clock.
2. Once the NVR is accessed and EN is asserted, output signal BUSY is also asserted. This signal stays high for a period of the non-volatile access delay. During this period, all the memory-related input ports of the non-volatile register must be kept constant.
3. The output data of the NVR (represented by the DOUT output of NVR) can be captured when BUSY is low.

NVRE implements a counter to count down from the time delay to trigger the BUSY signal. NVRE entity also outputs an extra signal (i.e., the BUSY_SIG signal), which is similar to BUSY, but pulled low one clock cycle before BUSY is pulled low. This signal can be used by synchronous processes to update the input ports of the NVR, in order not to demand extra clock cycles and operate continuously while BUSY is primarily intended for asynchronous circuitry.

4.1.2 Reset Block (RB). This entity fills the BRAM with zeros while the whole FPGA resets, i.e., the user pressed the hardware reset button. The RESET signal is asserted, which enables the reset block. The reason behind this module is that volatile register (BRAM block) contain old data after a real reset because the FPGA is not powered off. Hence this block wipes all memory to a initial defaults state. To achieve a complete memory wipe the RESET signal should be high for a number of clock cycles that equals the size of the non-volatile register to clear the whole BRAM.

4.1.3 Operation Consistency of NVR. The POWER_RESET can be triggered in the middle of an ongoing write operation to the NVR. In reality, a reset during a write operation to non-volatile memory (like Fe-RAM) does not leave non-volatile memory partially updated, i.e., either the word is written or not. In NORM, we followed a similar strategy to mitigate the side effects of power failures during NVR write operations. NORM guarantees that if the write operation is accepted by NVR (BUSY on), then the operation will be completed successfully (meaning that data is written).
4.2 Intermittency Emulator (IE)

Intermittency Emulator (IE), depicted in Figure 5, implements the auxiliary module presented in Section 3.1. This entity comprises a ROM memory to hold a voltage trace and a counter that iterates through the entries of the ROM. IE can also be prescaled (via the CLK DIVIDER block in Figure 5) to change trace duration (i.e., to slow down the iteration speed). Once the trace ends (the counter overflows), it restarts from the beginning. Inputs to this entity are THRESHOLD_VAL that is the set of desired voltage thresholds which will be compared against the current value in the ROM memory cell, and SELECT_THRESHOLD which is used to select the comparator whose output generates the desired POWER_RESET signal. Even at the synthesis level, IE can be configured to have multiple comparators allowing having more precise and granular control on the voltage status during runtime. Each comparator checks if the entry of the voltage trace (pointed by the counter) is below a given threshold value, every threshold is provided in advance by the user that can define both the quantity and the values. The multiplexer selects which of the provided thresholds is the one that triggers the POWER_RESET, the one that will be used by all entities of the volatile architecture as the main signal that resets the system due to a power failure. This entity also outputs THRESHOLD_COMP, which is a vector in which each bit indicates if voltage value is higher or lower than the corresponding threshold.

4.3 Energy Approximator (EA)

Energy Approximator (EA) entity is composed of a set of counters that are incremented by one at each clock cycle. The number of entities whose energy needs to be approximated determines the number of counters, which can be configured in the source code. Each counter expresses the energy consumption of an entity (during the time it is on) in terms of the clock cycles. The Instant Energy Calculator (IEC), explained shortly, computes the approximated energy consumption of the entities based on the number of clock cycles kept on the counter. It is worth mentioning that the more precise the parameters in Table 1 are, the more accurate the energy approximation is.

4.4 Instant Energy Calculator (IEC)

Instant Energy Calculator (IEC), whose implementation is depicted in Figure 6, converts the number of clock cycles held in the counters of EA into an energy value. IEC takes EA_VALUES_ARRAY
that holds all values of counters in EA and INDEX the offset for the previously mentioned array as an input to calculate the energy consumption of the corresponding entity. The calculation is triggered by the START_CALC signal. The energy consumption (per clock cycle) of all components is stored in ROM and defined by the user. It is worth mentioning that the energy consumption of the components can be obtained from their data sheets as well as by performing testbed measurements to observe the actual energy requirements. An internal finite state machine (FSM) manages all operations, like reading values from ROM and multiplication. An important point worth mentioning is that if a process runs long enough, the counters of EA can overflow. Consequently, larger registers and a bigger multiplier are required to hold the number of clock cycles and perform the calculation of the energy consumption. To remedy this issue, IEC calculates the approximated energy within a time interval by sampling EA counters at regular intervals and re-initializing the counters in EA. The outputs of IEC are ENERGY that holds the calculated energy and EVALUATION_READY that indicates that the calculation is finished. The ENERGY output can be accumulated in a shared memory location (e.g., DRAM) to further add up and process the approximated energy consumption.

5 EVALUATION OF NORM

In this section, we present our simulations, performed via Vivado Simulator [60], to understand how NORM can be leveraged to emulate a custom non-volatile logic (which keeps its state upon reset) together with a volatile logic (which loses its state upon reset). The details of the architecture implemented for our simulations and evaluation are given as follows.

5.1 Simulation Architecture

We implemented an architecture that comprises a series of three counters (which lose their values upon power failures), and a backup logic that implements a backup policy which regulates how frequently the system gets its state stored in non-volatile memory (a backup). This operation needs access to non-volatile registers (described in Section 4.1) hence the backup-logic regulates these transactions. The overall blocks forming the simulation architecture (together with NORM) and their connections are presented in Figure 7. Multiple backup policies can be implemented as different finite-state-machines in the backup logic block. Each backup policy leads to different simulation results of the emulated transiently-powered system since it changes the run-time behaviour. With
the exception of the non-volatile registers, all entities of the simulation architecture are thought as volatile, hence after an emulated power off, the components lose state. Simulation architecture implements store/recover operations of the counter registers into/from the non-volatile registers. Backup logic triggers these operations. Therefore, counters continue counting from where they left after an emulated power failure.

5.1.1 Volatile Counters. As depicted in Figure 7, the Volatile Counters block comprises a finite state machine (denoted as FSM), three array of flip-flops to hold counter values (denoted as FF), and three counter blocks that increment the values stored in the corresponding array of flip-flops (denoted as COUNTER). During normal operation each counter is increased sequentially and with different base values. Specifically, during the increment operation, the counter value is fetched from
Fig. 8. Voltage trace used in simulations—taken from [35]. This voltage trace depicts the voltage level of a capacitor that stores the harvested energy from an RFID reader. We averaged this trace in groups of 25 samples to reduce memory usage. The dotted line is the threshold used to simulate a shutdown event (POWER_RESET), which we set to 2.8 V. The total shutdown time is 75% of the trace.

Fig. 9. Comparison of counter 1 value, among the different policies.

Fig. 10. Comparison of the approximated energy consumption of all counters using the different policies.

the volatile memory first (i.e., the corresponding flip-flop array). Then, this value is incremented, and the result is saved in the same flip-flop array. Finally, the FSM block selects the next counter, and this operation is performed again, and so on. As mentioned previously, NORM emulates a power failure by setting the POWER_RESET signal. Since the FPGA is still on during this operation, the volatile counters do not lose their state. Therefore, FSM emulates a real memory reset process by clearing the dedicated array of flip-flops of the counters.
5.1.2 Backup Logic. The backup logic implements a finite state machine that calls store/recovery procedures that access the non-volatile register (NVR). The order and rate of these procedures define the backup policy implemented by the finite state machine. More precisely the logic can be tuned by means of an external parameter. We implemented three backup policies for our simulations:

1. **Dynamic Backup Policy (DBP):** DBP uses the output of $\text{IE}$ as a dynamic input parameter, i.e. the state of an internal comparator bounded to a voltage threshold (see comparators in 4.2). Hence the tuning parameter for DBP is the set of input voltage thresholds in $\text{IE}$. Thresholds are defined in advance by considering the characteristics of the emulated transiently-powered architecture and voltage trace. For simplicity, we used only one threshold to trigger the backup operation in our current DBP implementation. When the current voltage trace value drops below the given backup threshold value, DBP goes into the hazard mode to save the state of the volatile counters into NVR.

2. **Constant-time Backup Policy (CBP):** CBP uses a user-defined backup period to backup the counters periodically. This constant is the tuning parameter for this backup policy. A timer initialized with this value triggers the backup when the time runs out, then the cycle restarts. As in DBP the possible values the parameter can assume are limited by the architecture and the voltage trace.

3. **Task-based Backup Policy (TBP):** TBP backups the system on predefined computation boundaries. The policy tuning parameter is called backup task count and defines the necessary goal that the simulation architecture must reach before a backup can be issued. The time interval spanned by the voltage trace should be long enough to have at least one backup operation.

It is worth mentioning that the registers forming the finite state machine implemented by the Backup Logic are volatile and lose status after an emulated power failure. Therefore, the first operation that all finite state machines perform after a shutdown is the recovery procedure, this lets counters restore their values.

5.2 Simulation Results

We performed simulations concerning different backup policies to validate the architecture described in section 5.1. For each backup policy, we performed multiple runs of simulations with different parameter values, to understand the effect of these backup policies. In all simulation runs, we used the voltage trace depicted in Figure 8, which spans 100 microseconds with a system clock of 100 MHz. We set the access/request time of the NVR in NORM to 80 nanoseconds, which means that every process must wait for at least eight clock ticks to perform another request from the non-volatile memory. This value is compatible with similar non-volatile technologies like FeRAM [50].
5.2.1 Metrics. For each simulation run, we collected the following values:

1. **Counter Value**: This metric represents the value inside the counter 1 (depicted in Figure 7). Since the FSM increases the counters one at a time, each counter should wait that other counters get updated before increasing. Therefore, counter 1 is incremented with a frequency of 4.16 MHz under normal execution.

2. **Energy Approximation (Counters)**: This metric represents the approximated energy consumption (calculated by the EA) of all volatile counters depicted in Figure 7.

3. **Energy Approximation (NORM)**: This metric represents the approximated energy consumption of all components forming NORM.

5.2.2 Parameters. As mentioned previously, backup policies have different parameters, i.e., *backup threshold*, *backup period* and *backup task count*, respectively. In our simulations, we selected different values for these parameters. More precisely:

- **DBP backup threshold (mV)**: This value defines the point below which the architecture must stop and start performing a backup. It is compared against the voltage level of IE. While the voltage level is below the threshold, computation will not progress. The parameter starts from 3000 mV and is incremented with a step size of 10 mV until 5010 mV is reached.

- **CBP backup period (microseconds)**: This value defines a period used by an internal timer. When the timer fires, the architecture must perform a backup if there is still available energy. This parameter starts with 2 microseconds and is incremented with a step size of 2 \( \mu s \) until 398 microseconds are reached.

- **TBP backup task count (numerical value)**: This value defines a target goal that COUNTER1 must reach before the architecture can perform a backup. More precisely, when the counter value is a multiple of this value, the architecture stops and performs a backup. The range of possible values for *backup task count* is between 1 to 55.

5.2.3 Results. We run our simulations multiple times for each parameter value and using the same voltage trace and duration. Figure 9 presents the value of COUNTER 1 (counter1_val) considering three policies. This metric shows the amount of progress established (i.e., counter increment operation) despite the emulated power failures. Therefore, the greater the value of this metric is, the superior the backup policy is. During simulations, DBP exhibited the best case when the *backup threshold* equals 3040 mV. This is since DBP carries out backups only when necessary (following the voltage trend), without wasting time and execution. Moreover, the higher the backup threshold is, the lesser the available time for computation will be. Therefore, the counter1_val metric decreases with respect to the increased *backup threshold* value. On the contrary, CBP and TBP exhibited an irregular behavior concerning different parameter values. This situation occurs since *backup period* and *backup task count* are fixed, and they do not completely represent the dynamics of the voltage trace. The conclusion is that DBP is more responsive while the other two cannot adapt to the voltage trace dynamics.

Figure 10 presents the approximated energy consumption of the simulation architecture (including backup policy blocks) concerning different backup policies. Considering the parameter values that maximize the outcome of each policy from the previous simulation (counter1_val), the energy consumption of the DBP is the highest among the others. This is since the counters were on and incremented more with the DBP. One can calculate the amount of energy consumption...
per one counter increment operation as:

\[
E_{DBP} = \frac{1981}{223} = 8.88 \text{ Joules/Increment},
\]

\[
E_{CBP} = \frac{1768}{191} = 9.26 \text{ Joules/Increment},
\]

\[
E_{TBP} = \frac{1626}{184} = 8.84 \text{ Joules/Increment}.
\]

Therefore, from the energy consumption point of view, TBP is as good as the DBP for the particular voltage trace used in the simulations. Another observation is that the energy consumption trend presented in Figure 10 for CBP and TBP is the opposite of that of DBP. The reason is that increasing the \textit{backup threshold} in DBP reduces the available computation time, and in turn, decreases the available time for the counter increment operation. However, in CTB and TBP policies, increasing the \textit{backup period} and \textit{task count} parameters delays the backup, hence the volatile counters are incremented more before a backup operation.

Figure 11 presents the energy approximation of the NORM framework. The graphs in this figure are roughly following the opposite trend of the graphs depicted in Figure 10. This happens because when counters are active NORM is not, and vice versa. By considering the points where the tuning parameter maximize the COUNTER 1 value (pointed values in Fig. 11, one for each policy), it is possible to state that the lowest energy consumption, in term of NORM usage, is achieved by DBP. This checks with the theory since the principle of DBP is to use less frequently NVR and backup only when necessary.

As a summary of our simulations, we conclude that NORM can be used to validate and analyze logic systems, including non-volatile memory elements.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced NORM which is an FPGA-based emulation framework. NORM can emulate any intermittent computing system that exploits fast non-volatile memories to store temporary status in case of supply failures. NORM comprises auxiliary blocks that simulate the behavior of an unregular power supply, which is typical to any batteryless transient computing system powered by energy harvesters. NORM takes into account the delay of the NVMs and approximates the energy consumption of the emulated technology. Our evaluation showed that NORM can be used to emulate and validate a FeRAM-based custom non-volatile digital logic successfully. We conclude that NORM is appropriate for verifying the behavior of such new types of systems over long time scales, typical of duty-cycling energy-neutral Internet of Things (IoT) applications.

Future studies can target the emulation of a more sophisticated non-volatile logic, such as a non-volatile processor architecture. RISC-V processor family is freely available and a good candidate for IoT computing applications. A non-volatile RISC-V can be implemented, and its behavioral verification can be done using the proposed FPGA architecture. Moreover, the accuracy of the energy approximation block can be observed by comparing the actual ASIC implementation of the processor concerning its implementation in the proposed emulation architecture.

REFERENCES

[1] Tosiron Adegbija, Anita Rogacs, Chandrakant Patel, and Ann Gordon-Ross. 2018. Microprocessor optimizations for the Internet of Things: a survey. \textit{IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems} 37, 1 (2018), 7–20.

[2] Domenico Balsamo, Alex S Weddell, Anup Das, Alberto Rodriguez Arreola, Davide Brunelli, Bashir M Al-Hashimi, Geoff V Merrett, and Luca Benini. 2016. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. \textit{IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems} 35, 12 (2016), 1968–1980.
[3] Naveed Anwar Bhatti and Luca Mottola. 2017. HarvOS: Efficient code instrumentation for transiently-powered embedded sensing. In 2017 16th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN). ACM, 209–220.

[4] Alexei Colin and Brandon Lucia. 2016. Chain: Tasks and Channels for Reliable Intermittent Programs. In Proc. OOPSLA. ACM, Amsterdam, Netherlands, 514–530.

[5] Cypress. 2016. 1-Mbit (128K × 8) Quad SPI nvSRAM. https://www.farnell.com/datasheets/2280496.pdf Last accessed: Nov. 15, 2021.

[6] Cypress. 2018. 256-Kbit (32 K × 8) SPI nvSRAM with Real Time Clock. https://www.cypress.com/file/45216/download Last accessed: Nov. 15, 2021.

[7] Cypress. 2019. FM22L16 4-Mbit (256K × 16) F-RAM. https://www.cypress.com/file/136476/download Last accessed: Nov. 15, 2021.

[8] Cypress. 2021. Excelon Ferroelectric-RAM (F-RAM): The Lowest-Power Nonvolatile Memory. https://www.cypress.com/products/excelon-fram-memory#tabs-0-bottom_side-3 Last accessed: Nov. 15, 2021.

[9] Zhuohui Duan, Haikun Liu, Xiaofei Liao, and Hai Jin. 2018. HME: A lightweight emulator for hybrid memory. In 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 1375–1380.

[10] Fujitsu. 2021. Memory ReRAM, 8M (1024 K × 8) Bit SPI. https://www.fujitsu.com/jp/group/fsm/en/documents/products/teram/lineup/MB8SAS5MT-DS501-00060-2v1-E.pdf Last accessed: Nov. 15, 2021.

[11] Shyamnath Gollakota, Matthew S Reynolds, Joshua R Smith, and David J Wetherall. 2013. The emergence of RF-powered computing. Computer 47, 1 (2013), 32–39.

[12] Yizi Gu, Yongpan Liu, Yiqun Wang, Hehe Li, and Huazhong Yang. 2016. NVPsim: A simulator for architecture explorations of nonvolatile processors. In 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 147–152.

[13] Josiah Hester, Timothy Scott, and Jacob Sorber. 2014. Ekho: realistic and repeatable experimentation for tiny energy-harvesting sensors. In Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems. 330–331.

[14] Josiah Hester and Jacob Sorber. 2017. The Future of Sensing is Batteryless, Intermittent, and Awesome. In Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems. ACM, 21.

[15] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely execution on intermittently powered batteryless sensors. In Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems. 1–13.

[16] Matthew Hicks. 2017. Clank: Architectural support for intermittent computation. ACM SIGARCH Computer Architecture News 45, 2 (2017), 228–240.

[17] Hrishikesh Jayakumar, Arnab Raha, Woo Suk Lee, and Vijay Raghunathan. 2015. QuickRecall: A HW/SW approach for computing across power cycles in transiently powered computers. ACM Journal on Emerging Technologies in Computing Systems (JETC) 12, 1 (2015), 1–19.

[18] Vito Kortbeek, Kasim Sinan Yildirim, Abu Bakar, Jacob Sorber, Josiah Hester, and Przemyslaw Pawelczak. 2020. Time-sensitive Intermittent Computing Meets Legacy Software. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems. 85–99.

[19] Taemin Lee, Dongki Kim, Hyunsun Park, Sungjoo Yoo, and Sunggu Lee. 2014. FPGA-based prototyping systems for emerging memory technologies. In 2014 25nd IEEE International Symposium on Rapid System Prototyping. IEEE, 115–120.

[20] Xueqing Li, Sumitha George, Kaisheng Ma, Wei-Yu Tsai, Ahmedullah Aziz, John Sampson, Sumeet Kumar Gupta, Meng-Fan Chang, Yongpan Liu, Suman Datta, et al. 2017. Advancing nonvolatile computing with nonvolatile NCFET latches and flip-flops. IEEE Transactions on Circuits and Systems I: Regular Papers 64, 11 (2017), 2907–2919.

[21] Haikun Liu, Di Chen, Hai Jin, Xiaofei Liao, Bingsheng He, Kan Hu, and Yu Zhang. 2020. A Survey of Non-Volatile Main Memory Technologies: State-of-the-Arts, Practices, and Future Directions. Corr abs/2010.04406 (2020). arXiv:2010.04406 https://arxiv.org/abs/2010.04406

[22] Yongpan Liu, Zhibo Wang, Albert Lee, Fang Su, Chieh-Pu Lo, Zhe Yuan, Qi Wei, Yu Wang, Ya-Chin King, et al. 2016. 4.7 A 65nm ReRAM-enabled nonvolatile processor with 6X reduction in restore time and 4X higher clock frequency using adaptive data retention and self-write-termination nonvolatile logic. In Solid-State Circuits Conference (ISSCC), 2016 IEEE International. IEEE, 84–86.

[23] Brandon Lucia and Benjamin Ransford. 2015. A simpler, safer programming and execution model for intermittent systems. ACM SIGPLAN Notices 50, 6 (2015), 575–585.

[24] Kaisheng Ma, Xueqing Li, Karthik Swaminathan, Yang Zheng, Shuangchen Li, Yongpan Liu, Yuan Xie, John Jack Sampson, and Vijaykrishnan Narayanan. 2016. Nonvolatile processor architectures: Efficient, reliable progress with unstable power. IEEE Micro 36, 3 (2016), 72–83.

[25] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. 2015. Architecture exploration for ambient energy harvesting nonvolatile processors. In High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on. IEEE, 526–537.
[26] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2019. Alpaca: Intermittent execution without checkpoints. arXiv preprint arXiv:1909.06951 (2019).

[27] Kiwan Maeng and Brandon Lucia. 2018. Adaptive dynamic checkpointing for safe efficient intermittent computing. In 13th {USENIX} Symposium on Operating Systems Design and Implementation ({{OSDI\textsuperscript{18}}} ). 129–144.

[28] Kiwan Maeng and Brandon Lucia. 2020. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 1005–1021.

[29] Amjad Yousef Majid, Carlo Delle Donne, Kiwan Maeng, Alexei Colin, Kasim Sinan Yildirim, Brandon Lucia, and Przemyslaw Pawelczak. 2020. Dynamic task-based intermittent execution for energy-harvesting devices. ACM Transactions on Sensor Networks (TOSN) 16, 1 (2020), 1–24.

[30] Micron. 2005. P8P Parallel Phase Change Memory (PCM). https://www.digchip.com/datasheets/parts/datasheet/297/ NP8P128A13B6M60E.pdf. Last accessed: Nov. 15, 2021.

[31] Matteo Nardello, Harsh Desai, Davide Brunelli, and Brandon Lucia. 2019. Camaroptera: A Batteryless Long-Range Remote Visual Sensing System. In Proceedings of the 7th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems. ACM, 8–14.

[32] NORM. 2021. Github Repo. https://github.com/simoneruffini/NORM. Last accessed: Feb. 15, 2021.

[33] Texas Instruments. 2018. MSP430FR5969 LaunchPad Development Kit. http://www.ti.com/tool/MSP-EXP430FR5969 Last accessed: Nov. 15, 2021.

[34] Yu Omori and Keiji Kimura. 2019. Performance Evaluation on NVMM Emulator Employing Fine-Grain Delay Injection. In 2019 IEEE Non-Volatile Memory Systems and Applications Symposium (NVMSA). IEEE, 1–6.

[35] PersiSTLab. 2021. Voltage traces. https://github.com/PERSIStLab/BatterylessSim/blob/master/traces/README. Last accessed: Jan. 31, 2021.

[36] Matt Poremba and Yuan Xie. 2012. Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In 2012 IEEE Computer Society Annual Symposium on VLSI. IEEE, 392–397.

[37] Benjamin Ransford and Brandon Lucia. 2014. Nonvolatile memory is a broken time machine. In Proceedings of the workshop on Memory Systems Performance and Correctness. 1–3.

[38] Benjamin Ransford, Jacob Sorber, and Kevin Fu. 2012. Mementos: System support for long-running computation on {RFID}-scale devices. ACM Sigplan Notices 47, 4 (2012), 159–170.

[39] Emil Ruppel and Brandon Lucia. 2019. Transactional concurrency control for intermittent, energy-harvesting computing systems. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. 1085–1100.

[40] Alanson P Sample, Daniel J Yeager, Pauline S Powledge, Alexander V Maminshhev, Joshua R Smith, et al. 2008. Design of an {RFID}-based battery-free programmable sensing platform. IEEE transactions on instrumentation and measurement 57, 11 (2008), 2608.

[41] Rishad Shafik, Alex Yakovlev, and Shidhartha Das. 2018. Real-Power Computing. IEEE Trans. Comput. (2018).

[42] Xin Shi, Tongda Wu, Keni Qiu, Huazhong Yang, and Yongpan Liu. 2018. Time Stamp Based Scheduling for Energy Harvesting Systems with Hybrid Nonvolatile Hardware Support. In 2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 339–344.

[43] Sivert T. Sliper, William Wang, Nikos Nikoleris, Alex S. Weddell, and Geoff V. Merrett. 2020. Fused: Closed-Loop Performance and Energy Simulation of Embedded Systems. In 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 263–272.

[44] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. 2017. A Ferroelectric Nonvolatile Processor with 46um System-Level Wake-up Time and 14um Sleep Time for Energy Harvesting Applications. IEEE Transactions on Circuits and Systems I: Regular Papers 64, 3 (2017), 596–607.

[45] Fang Su, Kaisheng Ma, Xueqing Li, Tongda Wu, Yongpan Liu, and Vijaykrishnan Narayanan. 2017. Nonvolatile processors: Why is it trending?. In Proceedings of the Conference on Design, Automation & Test in Europe. European Design and Automation Association, 966–971.

[46] Xifan Tang, Edouard Giacomin, Baudouin Chauviere, Aurelien Alacchi, and Pierre-Emmanuel Gaillardon. 2020. OpenFPGA: An open-source framework for agile prototyping customizable FPGAs. IEEE Micro 40, 4 (2020), 41–48.

[47] Xifan Tang, Edouard Giacomin, Giovanni De Micheli, and Pierre-Emmanuel Gaillardon. 2018. Post-P&R performance and power analysis for {RRAM}-based FPGAs. IEEE Journal on Emerging and Selected Topics in Circuits and Systems 8, 3 (2018), 639–650.

[48] Avalanche Technology. 2021. Data Endurance, Retention and Field Immunity in STT-MRAM. https://www.avalanche-technology.com/wp-content/uploads/AN000002-Avalanche-STT-MRAM-Device-Characteristics-and-Capabilities.pdf Last accessed: Nov. 15, 2021.

[49] Texas Instruments. 2018. MSP430FR5969 LaunchPad Development Kit. http://www.ti.com/tool/MSP-EXP430FR5969 Last accessed: 2018.
[50] Texas Instruments, Inc. 2014. FRAM FAQs. https://www.ti.com/lit/pdf/slat151. Last accessed: 2020.
[51] Alessandro Torrisi, Davide Brunelli, and Kasim Sinan Yildirim. 2020. Zero Power Energy-Aware Communication for Transiently-Powered Sensing Systems. In Proceedings of the 8th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems. ACM, 43–49.
[52] Joel Van Der Woude and Matthew Hicks. 2016. Intermittent computation without hardware support or programmer intervention. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). 17–32.
[53] various authors. 2021. https://en.wikipedia.org/wiki/Ferroelectric_RAM Last accessed: Nov. 15, 2021.
[54] various authors. 2021. https://en.wikipedia.org/wiki/Magnetoresistive_RAM Last accessed: Nov. 15, 2021.
[55] various authors. 2021. https://en.wikipedia.org/wiki/Resistive_random-access_memory Last accessed: Nov. 15, 2021.
[56] various authors. 2021. https://en.wikipedia.org/wiki/Phase-change_memory Last accessed: Nov. 15, 2021.
[57] Yiqun Wang, Yongpan Liu, Shuangchen Li, Daming Zhang, Bo Zhao, Mei-Fang Chiang, Yanxin Yan, Baiko Sai, and Huazhong Yang. 2012. A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In ESSCIRC (ESSCIRC), 2012 Proceedings of the. IEEE, 149–152.
[58] Tongda Wu, Lefan Zhang, Huazhong Yang, and Yongpan Liu. 2018. An Extensible System Simulator for Intermittently-Powered Multiple-Peripheral IoT Devices. In Proceedings of the 6th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems. ACM, 1–6.
[59] Xilinx. 2021. Vivado Design Suite. https://www.xilinx.com/products/design-tools/vivado.html. Last accessed: Jan. 31, 2021.
[60] Xilinx. 2021. Vivado Simulator. https://www.xilinx.com/products/design-tools/vivado/simulator.html. Last accessed: Jan. 31, 2021.
[61] Xilinx. December 9, 2019. Block Memory Generator v8.4. https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf
[62] Kasım Sinan Yıldırım, Amjad Yousef Majid, Dimitris Fatoukas, Koen Schaper, Przemyslaw Pawelczak, and Josiah Hester. 2018. Ink: Reactive kernel for tiny batteryless sensors. In Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems. 41–53.
[63] Guoliang Zhu, Kai Lu, Xiaoping Wang, Xu Zhou, and Zhan Shi. 2017. Building emulation framework for non-volatile memory. {IEEE} Access 5 (2017), 21574–21584.