Fused: Closed-loop Performance and Energy Simulation of Embedded Systems

Sivert T. Sliper*, William Wang†, Nikos Nikoleris†, Alex S. Weddell* and Geoff V. Merrett*
*{s.sliper, a.weddell, g.merrett}@ecs.soton.ac.uk; †{william.wang, nikos.nikoleris}@arm.com
*School of Electronics and Computer Science, University of Southampton, UK; †Arm Research, Cambridge, UK

***Abstract*—Energy-driven computing is an emerging paradigm that aims to fuel the proliferation of tiny and low-cost IoT sensing and monitoring devices. Energy-driven computers are generally powered by energy harvesting sources, and adapt their operation at runtime according to energy availability; thus, they must be designed and tested according to the expected dynamics of their power source. However, today's processor simulators and debuggers typically assume that power is always available, so they are unable to correctly model the interactions between power supply, power consumption and energy-driven execution. To address this shortcoming, we propose *Fused*, an open source full-system simulator for energy-driven computers. *Fused* models execution, power consumption, and power supply in a closed loop, thus correctly models the interaction between them. It targets energy-driven embedded systems, and employs SystemC for digital and mixed-signal simulation to model a microcontroller and mixed-signal circuitry, enabling hardware-software codesign and design space exploration. *Fused* includes a high-level power modelling methodology, whereby events recorded during simulation are correlated to power measurements of real hardware to extract features for power modelling. Results show that *Fused* can model the execution time and power consumption of a commercially available microcontroller with a geometric mean error of 0.2% and 3.4% respectively, across a wide range of workloads. Through a case-study, we demonstrate that *Fused* can accurately model a state-of-the art intermittent computing system, where execution is heavily dependent on energy availability: although up to 70 power cycles were needed to complete the tested workload on the constrained energy supply, *Fused* modelled the completion time with less than 7% error.**

***Index Terms*—Virtual prototyping, SystemC, Embedded systems**

I. Introduction

With the advent of IoT, tiny, low-cost computing devices are becoming ubiquitous. The Ericsson mobility report predicts that by 2024, there will be 22 billion devices connected to the internet, excluding PCs, laptops, tablets and mobile phones [1]. This drives a need for ever smaller and more low-cost devices. Traditionally, this need has been met by battery-powered devices, but the batteries limit deployment lifetime and lead to increased cost, volume and mass. An attractive option is to augment, or even replace, battery power with energy harvested from ambient solar, mechanical (wind, vibration, motion), RF, or thermal energy sources. But harvested energy is often scarce, dynamic, and unpredictable [2].

All data supporting this study are openly available from the University of Southampton repository at https://doi.org/10.5258/SOTON/D1200. This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) under an iCASE award and Grant EP/P010164/1.

To cope with the dynamics of harvested energy, energy-driven computing [3], [4] is emerging as a computational paradigm where the dynamic availability of energy is a fundamental component throughout the system and application design process. The operation of energy-driven systems is often governed by energy availability, adapting to deliver the highest possible quality of service for the energy available during deployment.

One example of an energy-driven mode of operation is energy-neutral operation [2], where systems aim to adapt their energy consumption to match harvested energy over long periods, using moderate amounts of energy buffering in the form of rechargeable batteries or supercapacitors.

Another example is intermittent operation [5], where the system retains computational progress through power cycles. Under intermittent operation, systems opportunistically make forward progress whenever energy happens to be available; these systems operate correctly on unstable power supplies while requiring only a tiny amount of energy buffering (typically on the order of $10\,\mu J$) [6]–[9], so devices can be made exceptionally tiny and low-cost.

Energy-driven operation brings new research and development challenges. Being governed by energy availability, the operation of these systems becomes difficult to reason about, validate and debug [10]–[12]. Today's development, validation and debugging tools typically assume that energy is always available, rendering them impractical when targeting energy-driven operation. To overcome this, and as an enabler and accelerator for the research and development of energy-driven computing systems, we propose *Fused*[1], an open source mixed-signal SystemC virtual prototype that:

- simulates execution, power consumption, and power supply in a closed loop, thus enabling efficient hardware-software codesign and design space exploration in energy-driven computing;
- hosts a GDB server to interface with most software development environments;
- enables debugging functionality across power cycles, and the ability to freeze and step through the dynamic power/energy state in lockstep with execution;
- executes unmodified binaries to be deployed on real hardware, and can integrate existing CPU emulators with

[1]**Fu**ll-system **S**imulation of **E**nergy-**D**riven Computers. Source code is publicly available at https://www.arm.ecs.soton.ac.uk/technologies/fused/.

relatively little development effort;
- enables modelling of external circuitry, such as energy harvesters and power management circuits, improving repeatability in evaluation.

We anticipate that hardware-software codesign will become increasingly important for energy-driven computing in coming years, as a plethora new non-volatile memories (NVMs) are emerging [13]. Byte addressable NVM with low access energy is a key enabler for e.g. intermittent operation, where computational progress must be retained through frequent reboots. A simulation tool such as *Fused* is necessary to determine which of these NVMs to choose, and how best to utilise them.

The key contributions of this work are as follows:

- *Fused*, a simulation framework tailored towards modelling energy-driven computers, utilizing powerful SystemC and SystemC AMS models of computation to succinctly, flexibly and accurately model the interplay between program behaviour, digital hardware, and analogue power management circuits.
- An event-based power modelling methodology, leveraging timing-accurate simulation to correlate a small set of high-level events with the power consumption of a commercially available microcontroller.
- A case-study that demonstrate hardware-validated simulation of a state-of-the-art intermittent computing system, using *Fused*.

This paper begins with background and a review of related works (Section II), introduces our experimental platform used for evaluation (Section III), then gives an overview of *Fused*'s model architecture (Section IV), before describing its high-level event-based power modelling methodology (Section V). We then evaluate *Fused* through experiments (Section VI), and finally a case-study shows how it is able to accurately model a state-of-the-art intermittent system (Section VII).

II. Background and Related Works

Development tools. Prior works have highlighted the unique challenges that emerge in energy-driven, and particularly intermittent, systems [10]–[12]. *Ekho* [11] records IV-surfaces (current-voltage curves over time) of energy harvesters, so that they can be replayed in the lab for realistic and repeatable evaluation of energy-driven systems. *EDB*, the energy-interference-free debugger, [10] is a hardware debugging tool that aims to enable debugging of energy-driven systems without interfering with their energy state. Additionally, *EDB* proposes new debugging primitives, such as energy-breakpoints, to facilitate efficient debugging of intermittently-powered devices. These approaches are complimentary to *Fused*, and could be used in conjunction with *Fused* when hardware is available.

Power estimation. The typical workflow for early performance and power modelling of digital circuits consists of RTL design, synthesis and simulation to obtain accurate power and timing information. This workflow works well for continuously-powered circuits, and can yield very accurate results for a specific CMOS process node. It is, however, not suitable for an energy-driven computer, because it does not take into account the effect of energy availability and consumption on execution and vice versa. For example, under intermittent operation, a device commonly reboots as a result of severely constrained power supply, thus its execution flow cannot be determined without modelling energy availability. This workflow is also inflexible, very time consuming, and computationally expensive, hindering efficient design space exploration.

A popular option in high-performance computing (mobile, desktop and server) is to use *gem5* [14], where processing systems are modelled at a much higher level of abstraction. Here, software is written to model hardware behaviour, mainly by abstracting signals and operations to function calls. It is not synthesizable, hence not a replacement for RTL, but it provides for fast simulation and prototyping, allowing reasonably accurate power and performance modelling of complex systems running complex software [15]. Power consumption can be estimated in gem5 by recording architectural events and feeding them to a power modelling tool [15]. However, being optimised for simulation speed and for modelling high-performance systems, *gem5* lacks cycle-accurate capability, and is far too complex to efficiently and flexibly model low-power microcontrollers.

Simulating energy-driven computers. A key factor for energy-driven computers is that their behaviour is governed by the availability of energy. Their power consumption can even affect the amount of power being harvested, because of the non-linear IV curve of many energy harvesters [11]. This is not the case for traditional computers, hence *gem5* and RTL workflows can decouple power and performance modelling.

We postulate that, for energy-driven computers, power and performance must be modelled simultaneously, in a closed feedback loop. Furthermore, timing accuracy requirements are much stricter for energy-driven devices, because they might only be active for a few thousand clock cycles at a time [8].

NVPSim, a *gem5*-based model of the THU1010N [16] non-volatile processor (NVP)[2] was proposed in [17], and extended in [18] to include limited support for modelling peripherals. *NVPSim* supports only a fully custom in-house NVP based on an architecture rarely used in today's embedded systems, and furthermore assumes a fully non-volatile processor. Its extension for modelling peripherals also requires specialised driver code. The simulator therefore does not run the exact same binary as the real device, and may operate differently, potentially hiding hardware and/or software bugs.

Ma et al. [19] explored the microarchitecture of NVPs with RTL simulation, using *NVSim* [20] as the NVM model. The paper focused on comparing execution pipelines (non-pipelined, in-order, out-of-order), as well as exploring which parts of the microarchitectural state to save on power failure. Simulating the THU1010N, their RTL based simulations

[2]A non-volatile processor is a processor that employs non-volatile logic elements instead of, or as a backup for, regular volatile logic, so that it doesn't lose state when power fails.

achieved intermittent execution of several benchmarks with reported performance errors of less than $5\,\%$; however the simulation method is only briefly described. Notably, with a fully non-volatile processor, power and performance can more easily be calculated from an execution trace after simulation (decoupled power and performance), because the executed program can be agnostic to reboots.

Siren [21] extended the MSP430 simulator *MSPSim* [22] to include NVM, and basic energy simulation capabilities. *Siren*'s analog/energy domain consists of a capacitor model that also controls whether or not execution is active, and an *Ekho* emulator, that replays *Ekho* IV surfaces in simulation. *Siren* does not, however, realistically simulate energy consumption, because it assumes a single static energy consumption per instruction. In our power profiling (Section V), we show that the energy consumption of the MSP430 varies by more than $2\times$, depending on several factors, including memory allocation and access patterns.

Analytical models. Several analytical models have been proposed to evaluate intermittent computing methods [6], [23], [24] and energy-neutral systems [2]. The most comprehensive, *EH-model* [24], can provide early estimations of completion time for several recent intermittent computing methods using different state retention mechanisms.

Summary. Although there have been several prior works that target simulation of energy-driven/intermittent systems, they lack support for modelling external circuitry beyond a simple storage capacitor, and none include a method for capturing power modelling parameters. Most also lack the modularity and flexibility necessary to allow efficient hardware-software codesign. Prior works in analytical modelling of intermittent systems can be beneficial early in the design process, to guide simulation efforts.

III. Experimental Setup

The experimental platform used for the validation of *Fused* is a customised version of the MSP430FR5994 Launchpad Development Kit. A simplified schematic is shown in Fig. 1. It was modified in order to more efficiently support intermittent operation, and to provide high-bandwidth current measurements of the microcontroller (MCU). During the experiments performed for this work, the clock frequency of the microcontroller was set to $8\,\mathrm{MHz}$. The external circuitry consists of a $4.7\,\mu\mathrm{F}$ ceramic capacitor used for energy storage, a load switch that can disconnect the microcontroller from its supply voltage, and a low-power comparator with built-in voltage reference and hysteresis that monitors the supply voltage. The full schematic and PCB design files are published in the repository associated with this paper.

When charging the system from a cold start ($v_{cap} = 0\,\mathrm{V}$), the load switch remains open until the comparator closes it, i.e. once $v_{cap} > 3.5\,\mathrm{V}$. The microcontroller must then activate a pull up on the positive input of the comparator to keep the switch closed. If the GPIO pin is left in a high impedance state (High-Z), the comparator will open the switch at $v_{cap} = 3.4\,\mathrm{V}$, due to its $0.1\,\mathrm{V}$ built-in hysteresis. By

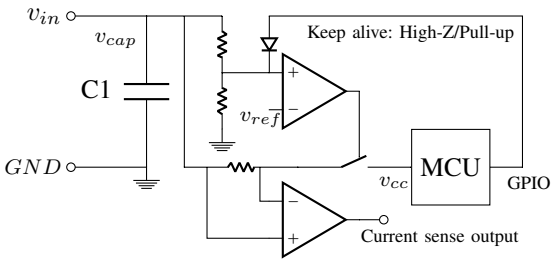

Fig. 1. Hardware test platform.

enabling pull-up on the GPIO pin, the microcontroller can keep the switch closed, and thus remain operational until it decides to open the switch to recharge C1, or until v_{cap} drops below the minimum operating voltage and the microcontroller browns out. The microcontroller's internal ADC is used to detect power failures, so that it can save execution context to support intermittent computing; the internal comparator in the microcontroller could also be used for this purpose. While the microcontroller is powered off, its GPIO pins are left in an undefined state; the diode in Fig. 1 prevents current from flowing into the GPIO pin in this situation.

The current sense amplifier converts milliamperes of current draw to volts ($1\,\mathrm{V/mA}$), and is used for gathering high-bandwidth ($\approx 100\,\mathrm{kHz}$) current traces of the microcontroller's current consumption. This capability is used to profile the microcontroller's current consumption, to train *Fused*'s power model, and for evaluation. The current sense amplifier is powered by a separate supply, so that its power consumption is excluded from measurements. An oscilloscope is used to measure the current sense output, and a logic analyser for monitoring microcontroller pins; both measurements were triggered by a common GPIO pin.

IV. Model overview

Simulation target. We target a typical embedded system, comprising a microcontroller, power management circuitry, and a power supply, as described in Section III. The proof-of-concept implementation of *Fused* targets the MSP430FR5994 microcontroller. This microcontroller has been widely used in the energy-driven computing community because its low-power, byte-addressable and non-volatile FRAM memory enables efficient state retention through power cycles. *Fused*'s microcontroller model implements a subset of the modules within an MSP430FR5994, including the CPU, a bus, memories, some internal peripherals, and some externally interfacing peripherals. An overview of *Fused* is shown in Fig. 2.

Requirements. The main requirement when modelling an energy-driven computer is that execution must be modelled simultaneously with power consumption and supply. Additionally, to aid in early design space exploration, the model needs to be flexible, and relatively fast.

Implementation. *Fused* is implemented using SystemC, an open C++ based IEEE-standardised language for designing and modelling digital electronic devices [25]. This brings the

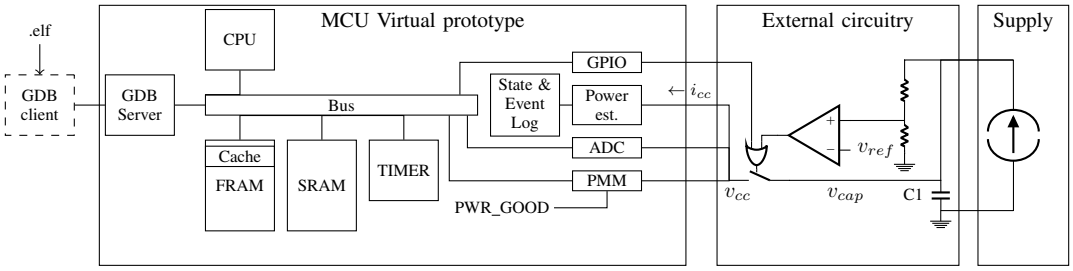

Fig. 2. Model architecture of Fused. This proof-of-concept implementation targets the system shown in Fig. 1, but all modules within the virtual prototype, external circuitry and supply can readily be modified or replaced.

advantage of combining several different models of computation to enable flexible and fast design space exploration, while allowing more detailed modelling where necessary.

Most of the modules within *Fused* are implemented using Transaction Level Modelling (TLM), a high-level fast and flexible modelling methodology. Where it is more appropriate, e.g. within peripherals and for interrupts, RTL-like modelling is used. Furthermore, by use of the SystemC Analogue and Mixed-Signal extensions (SystemC AMS) [26], complex power supplies and power management circuits can be modelled in a closed loop with execution.

GDB server. *Fused* hosts a GDB server that acts as the software interface to the model so that code can be debugged in the same way as on a hardware platform, using breakpoints, single-stepping, memory examination, and more.

Because *Fused* models power and execution in a closed loop, the analog circuitry also pauses when execution is halted by the debugger. Thus *Fused adds the capability of stepping through code in lockstep with the dynamic power supply and energy storage*. This is an essential feature, when researching intermittent computers that race to finish saving state before the stored energy runs out. Using monitor outputs, such as an execution trace linked to the supply voltage, the programmer can refine the application by iterative testing using *Fused*. When satisfactory results are achieved, the same application binary can be deployed onto the real device.

CPU. *Fused* includes CPU models for the *Cortex M0* and the *MSP430* instruction set architectures that execute unmodified binaries. All memory accesses from the CPU models go through the SystemC bus, and they use *Fused*'s API to consume simulation time. *Fused* is designed to be flexible, so that other microcontrollers and instruction set architectures can be modelled with minimal effort. We focus our evaluation on *Fused* on an *MSP430*-based microcontroller, because it is currently the most widely used platform in published works on intermittent computing due to its energy-efficient on-chip FRAM NVM.

Bus. The bus model is implemented using a blocking TLM interface, i.e. only a single transaction can be active at any time. This model corresponds well to a crossbar bus. For evaluation, the data and bus width are set to $16\,\mathrm{bit}$.

Peripherals. The bus communicates with peripherals using a TLM blocking interface. This means that all peripherals have a simple, common interface, so adding new peripherals requires minimal effort. Some peripherals have one or more interrupt request outputs, which are routed to the CPU via an interrupt arbiter. Externally facing peripherals have their ports routed to the top level microcontroller interface, so that they can interact with external digital and analog signals.

Power management module (PMM). The PMM monitors the supply voltage, and controls the core voltage within the microcontroller, turning it on when the supply voltage reaches approximately $1.88\,\mathrm{V}$, and turning it off when it drops below approximately $1.80\,\mathrm{V}$. *Fused* models this behaviour with a signal *PWR_GOOD*. While *PWR_GOOD* is asserted, execution continues, otherwise it is halted. A reset to default values for all internal registers and volatile memory is performed on the positive edge of *PWR_GOOD*.

Memory. The bus communicates with memory through a blocking TLM read/write interface, where the memory module annotates the transaction with access delay. The volatile byte-addressable SRAM memory has a simple single-cycle access delay. The non-volatile byte-addressable FRAM memory caches reads to reduce power consumption and average access latency. Its access time is nominally one clock cycle, but additional wait states on miss can be added by use of a control register; this is to avoid access time violations when running the CPU at clock speeds above $8\,\mathrm{MHz}$. The cache consists of four $64\,\mathrm{B}$ lines arranged in two sets. From experiments with the MSP430FR5994 (see Section VI), we found that writes go directly to FRAM, invalidating hits in the cache, and that the replacement policy is likely to be LRU (least recently used). Because the memory model in *Fused* is implemented in TLM, changing the bus parameters, memory sizes and delays, etc. requires minimal effort.

Event & state logging. *Fused* implements a global logger that records states and event rates at runtime. Each module registers its events during elaboration and reports the event on each occurrence. Similarly, they report their state at the beginning of simulation, and when it changes. Event counts within a configurable time step (e.g. $100\,\mu\mathrm{s}$) are then aggregated, logged, and reported to the power estimator.

Power estimator. The power estimator computes the current consumption of the microcontroller based on the state and event counts within the current time step. It estimates current consumption rather than power, because of the targeted plat-

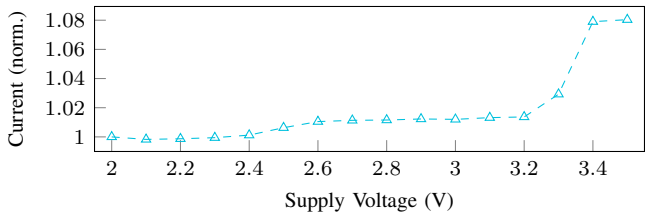

Fig. 3. Average current draw across all workloads as a function of supply voltage, normalised to the current draw at 2 V.

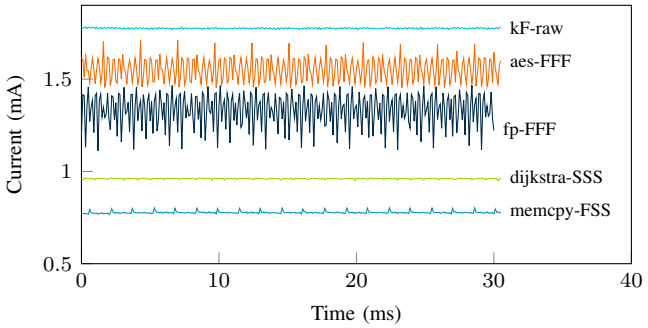

Fig. 4. A selection traces showing that current consumption is highly application-dependent, and can exhibit significant variations over time.

form's on-chip linear voltage regulator, which draws nearly constant current regardless of supply voltage. This current is then fed to the external circuitry.

External circuitry and power supply. Circuitry external to the microcontroller is modelled in the timed data flow (TDF) model of computation, i.e. essentially a set of equations that are evaluated according to a static schedule. For the purposes of this paper, the model includes a constant-current power supply model, also modelled in TDF, but it is trivial to exchange this with a different supply that e.g. models an energy harvester, or replays traces.

V. Power Modelling Methodology

This section describes the proposed methodology for profiling and modelling the energy consumption of real hardware platforms. This is useful both for modelling real systems, and for estimating the effect of changes to the hardware. The power model is modular, so if hardware is unavailable, the parameters can also be found from e.g. an RTL power estimation flow. The proposed methodology is similar to instruction-level power modelling (ILPM) [27], commonly used for high-level power modelling of microcontrollers. However the power model used herein is focussed on memory access energy, in place of per-instruction energy, because the target platform's energy consumption depends more strongly on memory accesses than on instructions. The authors expect the energy consumption of memories to be an increasingly important factor when developing microcontrollers that utilise emerging NVMs. Another key difference is that the model developed herein relies on far fewer parameters, and thus substantially decreases the burden on writing test programs.

The primary objective of the power model used for this demonstration of *Fused*, is to obtain an explainable model with a high degree of generality, and thereby avoiding overfitting (rather than demonstrating the highest possible accuracy). To this end, a minimal set of generic features are used, mainly based on memory accesses. Other users of *Fused*, may choose to include more numerous and detailed features to gain accuracy, especially if tailoring the model towards specific hardware or investigating the power consumption of specific instructions.

The power estimator of Fig. 2 estimates power consumption in the current time step based on module states and event-counts. To do that, it multiplies each event count with its energy consumption, and similarly sums the current consumption of each module state. The current consumption, i_{cc}, per timestep can be expressed as

$$i_{cc} = \left(\frac{\Sigma E_k c_k}{v_{core}\Delta t} + \Sigma I_{m,s}\right) \cdot c_{reg}(V),$$

where E_k is the energy consumption per occurrence of event k, c_k is the number of occurrences of event k within the current time step, v_{core} is the core supply voltage, Δt is the duration of the timestep, $I_{m,s}$ is the current consumption of module m in state s, and $c_{reg}(V)$ is a factor to compensate for current variation as a function of supply voltage. The first term within the parentheses converts dynamic energy (events) into current consumed at the core voltage, the second sums the current consumption of states. Note that the target platform demonstrated herein uses an internal linear regulator to convert the external supply voltage, v_{cc} to the internal supply voltage, v_{core}, and thus should ideally draw constant current regardless of supply voltage (within operating bounds). However, the current consumption increases significantly when v_{cc} approaches the maximum operating voltage of 3.6 V, as shown in Fig. 3. To compensate for this, the current consumption is multiplied by $c_{reg}(V)$, implemented as a lookup table of the sample points from Fig. 3.

Power profiling. To find the energy and current consumption attributable to each event and state, respectively, we propose a power profiling flow based on correlating current measurements from hardware with event and state logs from simulation. This method leverages timing accurate simulation to pinpoint how much power consumption to attribute to each event.

First, traces of the current consumption of all workloads are measured on real hardware. In this step, it can be beneficial to use a high sampling rate, effectively to obtain a large number of samples per workload, so that the model can be trained robustly with fewer workloads. Figure 4 shows three examples of current traces from three workloads chosen to demonstrate that the average current consumption varies substantially between applications, and, in some applications, also varies rapidly over time. Thus a model that assumes constant power consumption over time and across workloads, is bound to have large errors.

Then, the workloads are simulated to collect event and state logs. A GPIO pin is used to indicate the start of each iteration of the workloads, so that measured current traces and simulated logs can be synchronized temporally. These synchronized event logs and current traces are then passed on to a regression step, which estimates the energy consumption attributable to each state and event.

Feature selection and linear regression. *Fused* can record an arbitrary amount of events during simulation, but not all of these are useful for power modelling. To obtain an explainable and stable model with a high degree of generality, a minimal set of generic features was chosen:

- FRAM-RHIT: Read hits in the FRAM cache. These reads are served from the cache, without incurring FRAM accesses.
- FRAM-RMISS: Read misses in the FRAM cache. These cause a read from FRAM, loading a cache line, before serving the data.
- FRAM-W: Writes to FRAM. The written data goes directly to FRAM. If the data is present in the cache, the relevant cache line gets discarded.
- SRAM-RW: Total number of accesses to SRAM.
- EX: CPU execution cycles, here defined as cycles where the CPU does not fetch or store any data.

Fused also records the rate of occurrence of each instruction, addressing mode and more. Including these features may improve the accuracy of the power model, but would require a vast set of workloads to ensure that every instruction is exercised sufficiently.

For the final regression step, the non-negative least squares (NNLS) method is used to ensure non-negative values for all events and states, corresponding to physical intuition (events cannot generate energy). The correlation coefficient of each event k, $coeff_k$, is multiplied by v_{core} and the timestep between samples, Δt, to obtain E_k, the energy-consumption per occurrence of the event:

$$E_k = \Delta t \cdot v_{core} \cdot coeff_k.$$

Hardware boot. When powering up the microcontroller, energy is consumed before any code is executed. We observed that the current draw during hardware boot was similar across power cycles. Fig. 5 shows the boot-current trace averaged over 128 power cycles, measured by power-cycling the MCU while measuring the current consumption with an oscilloscope. To simulate the real hardware, *Fused*'s PMM replays the current trace of Fig. 5 during boot, and delays execution for its duration. Most simulators can safely ignore this boot current, because the systems they target rarely reboot; for energy-driven computers, however, reboots can be frequent, and may constitute a significant part of total energy consumption (demonstrated in Fig. 11).

VI. Experimental Validation

Computational kernels & benchmarks. To train the regression model, and evaluate simulation accuracy, a suite of computational kernels and benchmarks was assembled. The computational kernels are small assembler programs written to exercise specific features of the targeted MCU. These features include: write and read accesses to FRAM and SRAM, sparse and dense reads from FRAM (to exercise the cache), jumps, and copying data between memories. They are not necessarily representative of real workloads, but help in validating correct execution, and in training the power model.

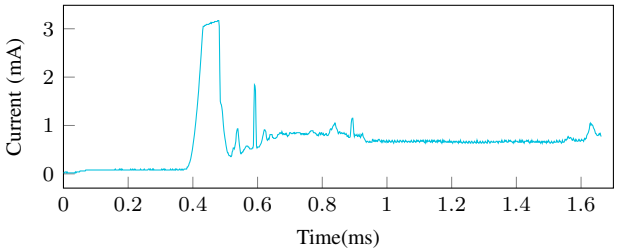

Fig. 5. Current consumption during hardware boot.

As representative workloads of embedded systems, several benchmarks from the BEEBS [28] online repository were ported. The benchmarks include common mathematical and data structure operations, and cryptographic workloads such as AES encryption, and SHA256 hashing. For brevity, we do not detail each benchmark, but the source code for all benchmarks is available online in the repository associated with this paper. A list of the workloads can be found in the horizontal axis of Fig. 8. For the remainder of this paper, the term "workloads" will be used when referring to both benchmarks and kernels.

Because the current consumption of the target platform depends heavily on memory allocation and access patterns, we generated several permutations of

$$\{CODE, DST\_DATA, SRC\_DATA\} \in \{FRAM, SRAM\},$$

i.e. permutations of allocating each of the three sections (lhs) to each of the memories (rhs). This is denoted in the benchmark names as CDS, where C is the code section, D is the destination data section, and S is the source data section. A kernel with its code allocated to SRAM, that copies data from FRAM to SRAM would thus get the suffix `SSF`. The stack (where used) was allocated to SRAM.

Execution time. To evaluate the simulator's execution time accuracy, the completion time of all workloads was simulated and measured on real hardware. A GPIO pin indicated the start and completion of each workload. The geometric mean error, across all benchmarks, between the simulated and measured completion time, was found to be $0.20\,\%$, and the maximum error $1.16\,\%$. The measured and simulated execution time of all benchmarks are available in the dataset associated with this paper, but are omitted here in the interest of brevity.

Cache model. To validate the cache model in *Fused*, the simulated cache miss rate was compared to the real hardware miss rate. Herein, we define the miss rate as being the ratio of cache misses over accesses, i.e. including both reads and writes. However, there are no direct ways of measuring miss rate on the real platform, so an indirect measurement was used.

To measure the miss rate,the execution time was measured while setting the FRAM cache miss penalty, WS, to 0 and to

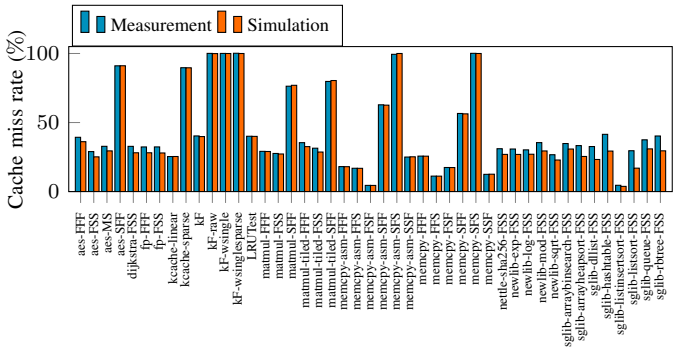

Fig. 6. Cache miss rate across benchmarks with more than 1000 total FRAM accesses.

15 clock cycles; setting $WS = 15$, causes the CPU to stall for 15 clock cycles on every cache miss, and hence the execution time becomes highly sensitive to the miss rate. The miss rate of a workload can then be calculated as

$$MR = \frac{f_{clk}(t_{WS15} - t_{WS0})}{15} \cdot \frac{1}{n_R + n_W}.$$

The first term calculates the total number of cache misses, and the second calculates the access rate. The clock frequency is denoted f_{clk}, t_{WS15} is the execution time when setting $WS = 15$, t_{WS0} the execution time when setting $WS = 0$, and $n_R + n_W$ is the number of read and write accesses to FRAM.

Figure 6 shows the measured and simulated miss rate. The geometric mean error between measurement and simulation is $2.69\,\%$. However, the model significantly underestimates the miss rate for certain programs. In the worst case, i.e. for *sglib-listsort-FSS*, the measured miss rate was $30\,\%$, but the simulated miss rate was only $17\,\%$.

The cache in the MSP430FR5994 is sparsely described in documentation. The cache is specified as a two-way set associative cache with a total of four lines holding 64 bits each. However, the documentation does not declare which replacement policy and write policy is used.

To ascertain which write policy is used, we wrote a micro benchmark, *kF-raw*, that repeatedly writes to and then reads from the same location in FRAM. The cache miss rate was found to be $100\,\%$, meaning that: on a write to FRAM, if the destination data already exists in the cache, it gets invalidated rather than updated. This is often referred to as a write-around policy.

The read policy of the cache is not specified in documentation, so we simulated LRU (least recently used), FIFO and LFU (least frequently used) policies and picked the one that best correlated to measurements. Additionally, we wrote a micro-kernel, *LRUTest*, that is designed to be sensitive to whether the replacement policy is LRU or FIFO (the two most likely candidates). The measurements from the LRU test, and from correlation indicate that the replacement policy is LRU.

The results of this analysis of cache miss rates indicate that the cache used in the MSP430FR5994 does implement write-around write policy, and a LRU-like read policy, but that there is additional undocumented behaviour during reads, that our cache model does not cover.

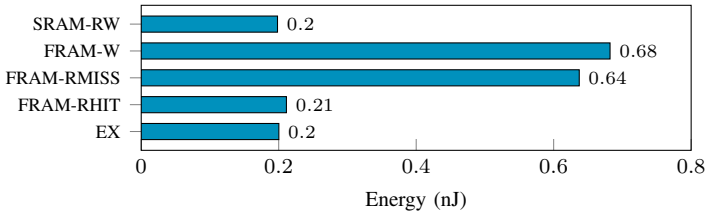

Fig. 7. Energy consumption per occurrence of events included in the power model.

Power estimation. To train and evaluate *Fused*'s power model, we ran a set of 60 workloads on the hardware platform, and recorded $30\,\mathrm{ms}$ current traces, starting from the beginning of each workload (as indicated by a GPIO pin). The measured traces were sampled at $1.25\,\mathrm{MHz}$, and averaged over 128 runs of each workload. Simulations were performed with a timestep of $100\,\mu\mathrm{s}$. The measured traces were then downsampled by averaging, to match the simulation sample rate, before the correlation step. The final sampling rate for the training data was set to $100\,\mu\mathrm{s}$ to allow some degree of temporal misalignment between measurement and simulation.

The estimated energy of each event, obtained through NNLS, is shown in Fig. 7. Accesses to FRAM are in excess of $3\times$ more costly in energy than those that can be served by the FRAM cache, or SRAM. This implies that, without knowing the cache miss rate of specific applications, power estimates will be severely inaccurate.

Figure 8 shows the measured average current of each workload on the left bars, and their estimated average current on the right bars. Current is used in place of power because of the targeted platform's on-chip linear voltage regulator. To evaluate the power model for unseen workloads, the set of workloads were divided into a training and a test set.

The bars for the estimated current also show the contributions attributable to each power model event. In general, FRAM-RMISS and FRAM-W are the strongest two predictors of current consumption, as expected from the device data sheet. The kernels *kF-raw* and *kcache-sparse* are both designed to have very high rates of access to FRAM; the first repeatedly reads and writes to the same FRAM-allocated variable (to test the write policy), the second consists entirely of long jumps that force a cache miss every second clock cycle. On the low-current end are workloads which operate mostly out of SRAM, or have very low miss rates in FRAM. Across all benchmarks, the geometric mean error of the current estimates is $3.4\,\%$, and the maximum error is $23.0\,\%$.

VII. Case-study: Simulating intermittent computing systems

To demonstrate *Fused*'s ability to simulate truly energy-driven systems, we simulate a state-of-the art reactive intermittent computing system [8] running AES encryption, and compare it to real hardware.

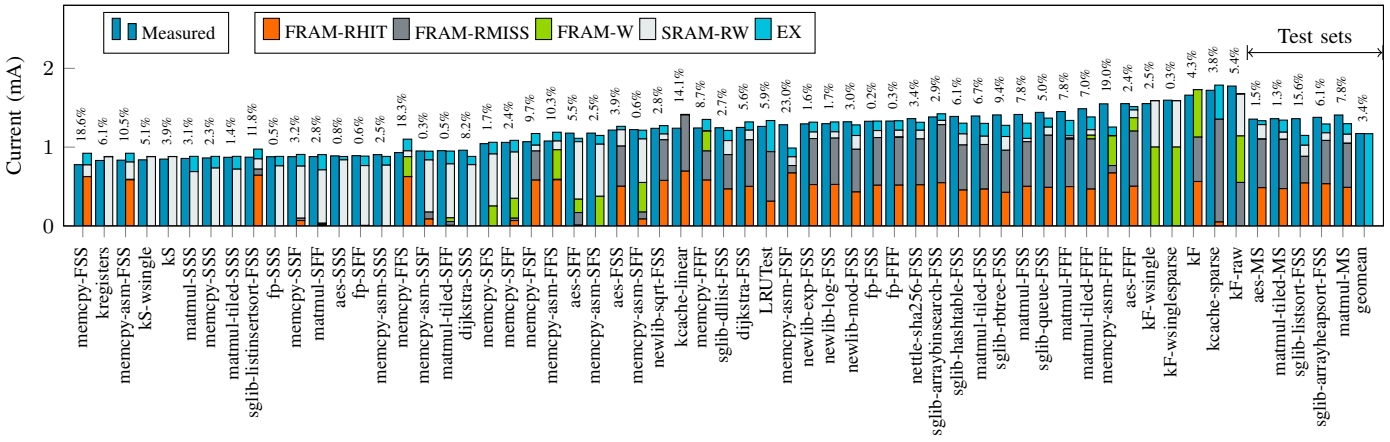

Fig. 8. Measured (left bars) and predicted (right bars) current consumption across workloads, sorted according to measured current consumption. The stacked predicted current breaks down the contributions of each feature. The number above each bar denotes the absolute percentage simulation error.

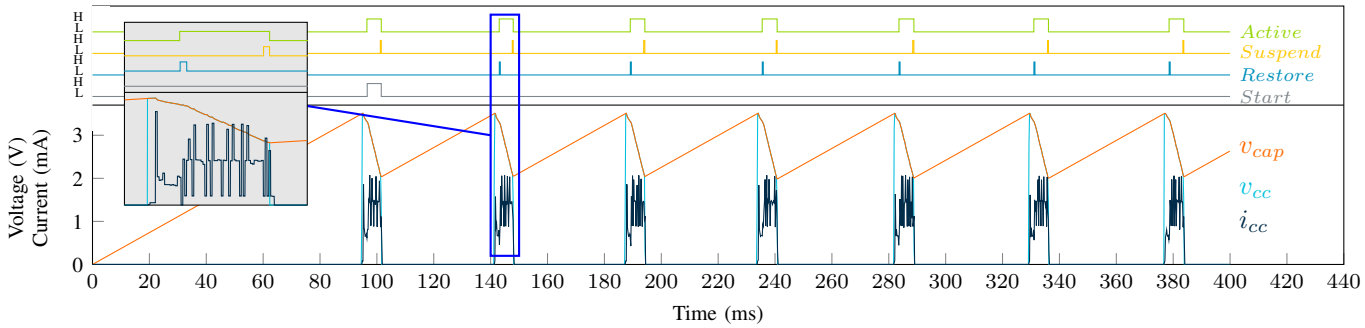

Fig. 9. Simulation trace of a reactive intermittent computing system powered by a 200 µA current-limited power supply. The top traces show the logic levels of GPIO pins indicating the operation of the device, and the lower traces show the microcontroller supply voltage (v_{cc}), the storage capacitor voltage (v_{cap}) and the current draw (i_{cc}).

Reactive intermittent computing. Intermittent computing systems (ICSs) use NVM to retain computational progress through power cycles, enabling them to execute applications opportunistically whenever energy is available. Reactive ICSs [6], [8], [29]–[31] use supply voltage monitoring to detect imminent power failures. When the supply voltage drops below a threshold, they *suspend* their volatile state, i.e. save volatile execution context to NVM, before brown-out. Then, when power returns, they *restore* context from the previously saved snapshot, and continue execution exactly where they left off. To be able to suspend state before brown-out, they require a tiny amount of energy buffering. In microcontrollers with low-power FRAM memory, the capacitance already present for power supply conditioning is usually sufficient to save a snapshot state [6], [8], [29], [30], but may fail if the snapshot is too large relative to the buffered energy. As an example, [8] found that their system could safely save only up to 4 kB on their platform which had 10 µF capacitance. Using *Fused*, one can step through these critical operations in lockstep with the energy-state to debug and validate correct operation.

In this case study, we evaluate *Fused*'s ability to model intermittent systems with *ManagedState* [8]. While many intermittent systems save the entirety of allocated volatile memory when saving snapshots [5], [6], [29], *ManagedState* divides memory into pages, and keeps track of active and modified pages, so that it can load pages on demand, and suspend only pages that are modified. In the context of simulation, an important aspect of *ManagedState* is that, unlike most reactive IC methods, the time and energy consumption of the *suspend* and *restore* operations vary during runtime based on how much state needs to be saved and restored. Furthermore, *ManagedState* adapts its suspend threshold at runtime, depending on the number of pages that need to be saved during the next *suspend* (the adaptive restore threshold was disabled during these experiments, because the external comparator has a constant restore threshold). These complications necessitate simulation over analytical methods.

Experimental setup. The completion time of a workload that encrypts a 2 kB string using AES was measured on the hardware platform shown in Fig. 1. Power was supplied by a current-limited power source connected to v_{in} and GND. This power source setup resembles that of a system connected to a solar harvester, but under more controlled conditions for the purposes of this demonstration. The current limit was varied from 200 µA to 2 mA in 100 µA steps. To avoid damaging the hardware, the power supply output voltage was limited

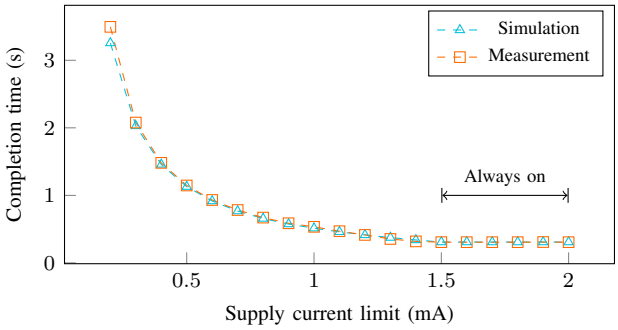

Fig. 10. Completion time of AES encryption when running intermittently, powered by a current-limited power source.

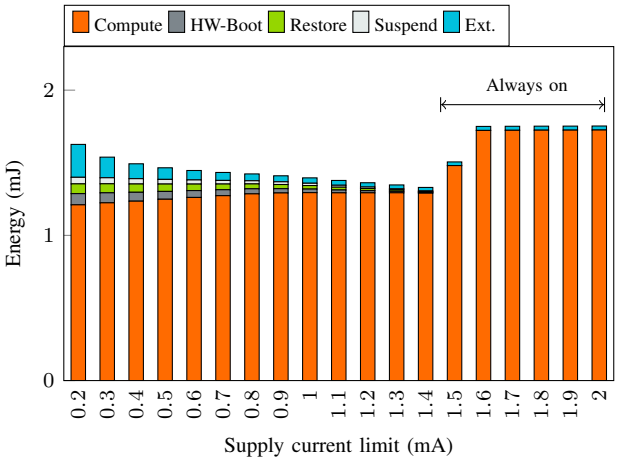

Fig. 11. Full-system energy consumption when running AES encryption intermittently, powered by current-limited power source. The energy consumption is divided into stacked bars for the external circuitry (*ext.*), hardware boot (*HW-Boot*), and the operational phases *restore*, *compute* and *suspend*.

to 3.59 V. For the following experiments, *ManagedState* was configured to assert a pull-up on the "keep alive" GPIO pin early in the boot process, and to de-assert it at the completion of *suspend*.

Results & analysis. Figure 9 shows a sample trace from *Fused*, where GPIO pins indicate when the system was active, the start of the benchmark, when a previous snapshot was restored, and when state was suspended.

Figure 10 shows the measured and simulated completion time of the application, for a range of supply currents, measured as the duration between positive edges of the *start* signal. The completion time increases exponentially with a linear decrease in supply current, mostly due to increased recharging time: as the supply current decreases, on-periods become shorter, and charging periods longer. The maximum error between simulation and measurement was 6.8 %, at the lowest-current measurement point. The mean absolute error was found to be 2.2 %. As indicated on the figure, the platform is continuously on for input currents larger than 1.5 mA.

Figure 11 shows the simulated energy consumption of the full system, calculated from the simulated current and voltage traces, and divided into components. The energy denoted "compute" is calculated as the energy consumption of the MCU while it is active, but not restoring or suspending. The increase in energy consumption at supply current limits exceeding 1.4 mA is caused by an increase in the average supply voltage; beyond 1.6 mA, the power supply is voltage-limited to 3.59 V. The compute energy remains relatively constant for supply current limits below 1.4 mA because the average supply voltage remains nearly constant; this, in turn, is because the comparator has a fixed on-threshold, and *ManagedState* adjusts the *suspend* threshold so that the voltage at the completion of *suspend* remains nearly constant. Hence, the voltage waveform seen in Fig. 9 remains similar, although temporally stretched/compressed.

The energy overhead of hardware-boot, *suspend* and *restore* grows linearly with the number of power cycles required to complete the workload. Even for the lowest-current measurement point, where one iteration of the workload spans an average of 70 power cycles, the combined energy overhead of *restore* and *suspend* constitute only 6.5 %.

The energy consumption of the external circuitry, however, increases proportionally with total completion time, because it is always powered on. These simulation results indicate that, when the supply current is weak, reducing the current consumption of the external circuitry would likely have a larger impact than further optimisation of the *suspend* and *restore* operations.

VIII. Conclusions

This paper presented *Fused*, a full-system simulator for energy-driven computers. Its focus is on closed-loop energy and performance simulation, as well as providing the flexibility needed to explore new hardware and software designs to improve energy-driven computers. Using *Fused*, a developer can rapidly get an accurate picture of the interplay between analog circuitry, digital hardware, and software.

Fused models execution time with a maximum error of 1.16 % across a broad set of 60 workloads. The power model of *Fused* profiles current consumption of real hardware, and correlates it to simulation events; this resulted in a power model using only five parameters, that achieves geometric mean error of 3.4 %, with a maximum error of 23.0 % across the 60 workloads. To evaluate *Fused*'s ability to model truly energy-driven systems, a case study simulated a state-of-the-art intermittent computing system, and validated results against real hardware. *Fused* modelled the completion time of an application running intermittently with a mean and maximum absolute error of 2.2 % and 6.8 %, respectively.

Our current research includes integrating more common on-chip peripherals to broaden *Fused*'s application area, and to explore the impact of emerging non-volatile memories on energy-driven computing. Building on prior works, we plan to integrate energy harvester models and traces, to aid in the development and repeatable evaluation of energy-driven devices.

REFERENCES

[1] Ericsson, "Ericsson Mobility Report June 2019," Tech. Rep., 2019.

[2] A. Kansal, J. Hsu, S. Zahedi, and M. B. Srivastava, "Power management in energy harvesting sensor networks," *ACM Transactions on Embedded Computing Systems*, vol. 6, no. 4, pp. 32–es, Sep. 2007.

[3] 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*. IEEE, May 2017.

[4] S. T. Sliper, O. Cetinkaya, A. S. Weddell, B. Al-Hashimi, and G. V. Merrett, "Energy-driven computing," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 378, no. 2164, p. 20190158, Feb. 2020.

[5] B. Ransford, J. Sorber, and K. Fu, "Mementos: System Support for Long-running Computation on RFID-scale Devices," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: ACM, 2011, pp. 159–170.

[6] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini, "Hibernus: Sustaining Computation During Intermittent Supply for Energy-Harvesting Systems," *IEEE Embedded Systems Letters*, vol. 7, no. 1, pp. 15–18, Mar. 2015.

[7] B. Lucia and B. Ransford, "A Simpler, Safer Programming and Execution Model for Intermittent Systems," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '15. Portland, OR, USA: ACM, 2015, pp. 575–585.

[8] S. T. Sliper, D. Balsamo, N. Nikoleris, W. Wang, A. S. Weddell, and G. V. Merrett, "Efficient State Retention Through Paged Memory Management for Reactive Transient Computing," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. Las Vegas, NV, USA: ACM, 2019, pp. 26:1–26:6.

[9] K. Maeng, A. Colin, and B. Lucia, "Alpaca: Intermittent Execution Without Checkpoints," *Proc. ACM Program. Lang.*, vol. 1, no. OOPSLA, pp. 96:1–96:30, Oct. 2017.

[10] A. Colin, G. Harvey, A. P. Sample, and B. Lucia, "An Energy-Aware Debugger for Intermittently Powered Systems," *IEEE Micro*, vol. 37, no. 3, pp. 116–125, 2017.

[11] J. Hester, L. Sitanayah, T. Scott, and J. Sorber, "Realistic and Repeatable Emulation of Energy Harvesting Environments," *ACM Transactions on Sensor Networks*, vol. 13, no. 2, pp. 1–33, Apr. 2017.

[12] A. Maioli, L. Mottola, M. H. Alizai, and J. H. Siddiqui, "On intermittence bugs in the battery-less internet of things (WIP paper)," in *Proceedings of the 20th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems - LCTES 2019*. Phoenix, AZ, USA: ACM Press, 2019, pp. 203–207.

[13] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electronics*, vol. 125, pp. 25–38, Nov. 2016.

[14] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, May 2011.

[15] M. Walker, S. Bischoff, S. Diestelhorst, G. Merrett, and B. Al-Hashimi, "Hardware-Validated CPU Performance and Energy Modelling," in *2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. Belfast: IEEE, Apr. 2018, pp. 44–53.

[16] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, M.-F. Chiang, Y. Yan, B. Sai, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *2012 Proceedings of the ESSCIRC (ESSCIRC)*. Bordeaux, France: IEEE, Sep. 2012, pp. 149–152.

[17] Yizi Gu, Y. Liu, Y. Wang, H. Li, and H. Yang, "NVPsim: A simulator for architecture explorations of nonvolatile processors," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. Macao, Macao: IEEE, Jan. 2016, pp. 147–152.

[18] T. Wu, L. Zhang, H. Yang, and Y. Liu, "An Extensible System Simulator for Intermittently-powered Multiple-peripheral IoT Devices," in *Proceedings of the 6th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, ser. ENSsys '18. Shenzhen, China: ACM, 2018, pp. 1–6.

[19] 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 *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. Burlingame, CA, USA: IEEE, Feb. 2015, pp. 526–537.

[20] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, Jul. 2012.

[21] M. Furlong, J. Hester, K. Storer, and J. Sorber, "Realistic Simulation for Tiny Batteryless Sensors," in *Proceedings of the 4th International Workshop on Energy Harvesting and Energy-Neutral Sensing Systems - ENSsys'16*. Stanford, CA, USA: ACM Press, 2016, pp. 23–26.

[22] J. Eriksson, A. Dunkels, and N. Finne, "Poster Abstract: MSPsim – an Extensible Simulator for MSP430-equipped Sensor Boards," in *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo Session*, Delft, Netherlands, 2007, p. 2.

[23] A. Rodriguez Arreola, D. Balsamo, A. K. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, and G. V. Merrett, "Approaches to Transient Computing for Energy Harvesting Systems: A Quantitative Evaluation," in *Proceedings of the 3rd International Workshop on Energy Harvesting & Energy Neutral Sensing Systems*, ser. ENSsys '15. Seoul, South Korea: ACM, 2015, pp. 3–8.

[24] J. San Miguel, K. Ganesan, M. Badr, C. Xia, R. Li, H. Hsiao, and N. Enright Jerger, "The EH Model: Early Design Space Exploration of Intermittent Processor Architectures," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Fukuoka, Japan: IEEE, Oct. 2018, pp. 600–612.

[25] S. A. IEEE, "IEEE Standard for Standard SystemC Language Reference Manual," Sep. 2011.

[26] IEEE, "IEEE Standard for Standard SystemC(R) Analog/Mixed-Signal Extensions Language Reference Manual," Apr. 2016, 10.1109/IEEESTD.2016.7448795.

[27] C. Chakrabarti and D. Gaitonde, "Instruction level power model of microcontrollers," in *1999 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 1, May 1999, pp. 76–79 vol.1.

[28] J. Pallister, S. Hollis, and J. Bennett, "BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms," *arXiv:1308.5174 [cs]*, Aug. 2013.

[29] 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.

[30] H. Jayakumar, A. Raha, and V. Raghunathan, "QUICKRECALL: A Low Overhead HW/SW Approach for Enabling Computations across Power Cycles in Transiently Powered Computers," in *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*, Jan. 2014, pp. 330–335.

[31] K. Maeng and B. Lucia, "Supporting Peripherals in Intermittent Systems with Just-in-time Checkpoints," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. Phoenix, AZ, USA: ACM, 2019, pp. 1101–1116.