An Energy Behavior Supporting Simulator for Nonvolatile Processor Based Self-powered System.

ABSTRACT

Battery-less devices and energy harvesting techniques are more and more attractive in today's IoT area. However, the intermittent and low-intensity ambient power sources cause insecure quality of service (QoS) of these self-powered IoT applications, which makes it essential to pre-simulate the QoS of an self-powered system and its energy managing strategy under various power conditions. Unfortunately, existing simulator lacks of supports on peripherals in a self-powered system and requires great efforts to implement the hardware-based energy managing schemes, such as NVP, DVFS and near-threshold designs.

This paper proposes EBeSS, an system-level simulator based on GEM5, supporting easy-configurable energy managing strategies on both processor and peripherals. EBeSS develops a configurable simulation object, virtual device, to support the functional and energy-related behaviors of peripherals, and energy message handling framework to support configured energy managing strategies. A real hardware prototype validates that the error of EBeSS is less that xx%. And a self-powered system design flow is given to reveal the potential of EBeSS on exploring the design space of parameters of a self-powered system (such as capacitor and energy managing scheme) with optimal performance.

1 INTRODUCTION

Battery-less devices are more and more popular in today's IoT area for its light-weight charge-free energy supply system and long lifetime. However, the energy supply quality of these devices is insecure according to different energy sources and different working environment. For instance, solar-power is the most popular ambient energy source, whose power supply intensity various within orders of microwatt to milliwatt according to the time, season, weather condition, altitude, location (indoor/outdoor) and etc. Therefore, developers need to evaluate the quality of service (QoS) of a self-powered system under specific power supply condition before developing an energy harvesting application. Without proper software simulator, developers have to implement a hardware prototype with appropriate energy managing strategies to realize evaluate quality of service (QoS), which costs unacceptable efforts and time.

To tackle this problem, previous works develop different simulation strategies for different types of self-powered devices. Target the systems with pure software energy managing and scheduling algorithms, EDB [1] proposes an energy supporting debugger to simulate the performance of intermittent powered devices. For the devices with hardware optimizations, such nonvolatile processor (NVP) [5, 8–10], NVIO [4], NVRF [11], and near-threshold circuits [], Ma et al. [6] provides a simulating strategy based on verilog source code to process register-transfer level (RTL) evaluation. Furthermore, Gu et al. [3] models the architecture and energy managing behaviors of NVP [9] and proposes NVPsim, a simulator based on GEM5 to explore the processor-level parameter design space of NVP.

Although provided usable simulation approaches for processor and software energy managing algorithms, the above mentioned simulation tools has two main draw backs. Firstly, these simulators cannot support the functional and energy-related behaviors of peripherals. Peripherals, such as sensors, radio transceivers and accelerators, are playing irreplaceable roles in IoT area by taking charge of the interconnections between the system and the environment, and improve the QoS. With intermittent power supply, these external hardware modules may cause inconsistency problems and needs to be safely recovered after sudden outages. Secondly, neither RTL-level simulation [6] nor NVPsim [3] supports flexible and easy configured hardware energy managing techniques such as NVP [9] and dynamic voltage-frequency scaling (DVFS) [2].

Focus on these problems, this paper proposes EBeSS, an **E**nergy **Be**havior **S**upporting **S**ystem-level simulator to explore the design space of a NVP based self-powered device. The contributions are listed as follows,

- EBeSS provides a configurable virtual device module to support the different kinds of peripherals;
- EBeSS realizes flexible and friendly hardware energy managing behavior supports with the help of the energy message handling (EMH) framework to realize and manage the energy-related behaviors of each hardware module;
- With EBeSS, we provides an example of self-powered system developing and optimizing procedure by exploring the design space of capacitor and energy managing strategy selection with different power traces and benchmarks. The analysis reveals an optimal design edge to improve the performance of self-powered systems.

The rest of the paper is organized as follows. Section 2 proposes the architecture of the simulator and introduces the two techniques used to support the energy behavior exploration of NVP and peripherals. Section 4 explores the performance of different energy behaviors with various power supply profile and application types.

2 EBESS DESIGN METHODOLOGY

This section demonstrates the design methodology of EBeSS. After introducing the overview of EBeSS structure, we illustrates the two key techniques embedded in the simulator. *Virtual Device* is a configurable module used to simulate the behaviors of peripherals. *Energy Message Handling Framework (EMHF)* provides an energy message interacting structure to manage the energy-related behaviors of each hardware module according to the energy managing strategies.

2.1 Structure Overview of EBeSS

EBeSS is an GEM5-based system-level simulator for self-powered systems. EBeSS adopts the fundamental usage and logic of kernel architecture in GEM5 and develops two novel components to support

DAC'2018, May 2018, San Francisco, CA, US
.

peripherals and energy management. EBeSS enables the ability of simulating the energy-related behaviors of self-powered systems, such as non-volatile processors, multi-frequency systems, and multi-voltage-domain systems. The system architecture of EBeSS is shown in Figure 1.

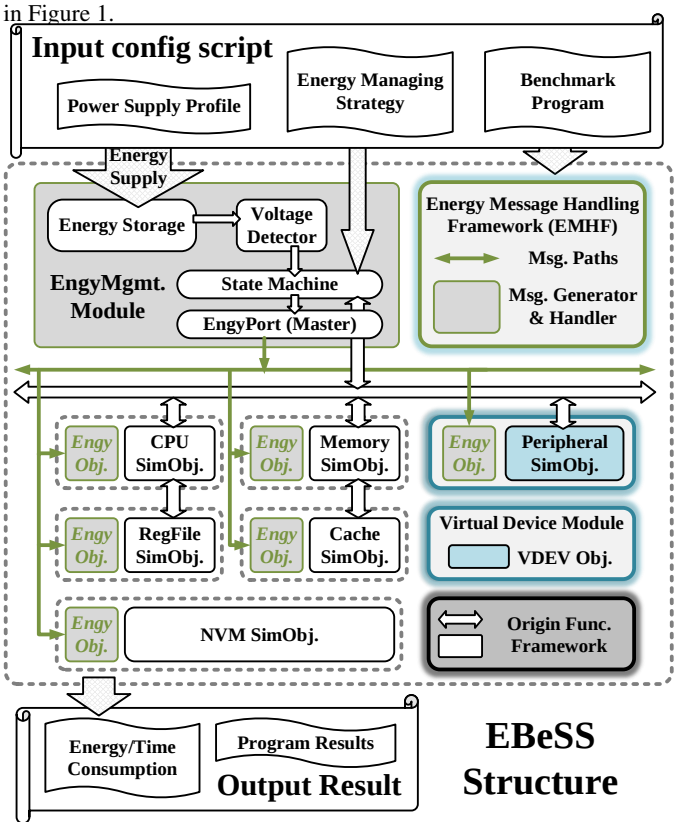

Figure 1: Simulator structure of EBeSS. Two new components are added based on original GEM5 architecture. EMHF generates and handles the energy messages according to energy conditions. The virtual device module creates an configurable object that supports the functionary and energy-related behaviors of peripherals.

To fill the blank of lacking peripheral supports, EBeSS adopts a new simulation object, *virtual device*, which defines the connections and functionary behaviors of various peripherals, as shown in Figure 1. Virtual device is a configurable, pluggable module connected the system bus, which can be accessed by processor module via virtual addresses. Virtual device supports the fundamental operations of a peripheral, including the peripheral register read/write, initialization, activation and interrupts. Details of the virtual device module is discussed in Section 2.2.

EMHF provides a message-based energy managing framework to support configurable energy managing strategies, as shown in Figure 1. EMHF contains three components, energy management module (EMM), energy ports and energy objects. EMM contains a state machine to generate the state conversion according to user's energy managing strategy. Energy ports define the energy message transmission rules and manage the interconnections between EMM and other hardware components. For each hardware component, EMHF reconstructs it with a new energy-related object, energy object (EnergyObject) allowing energy behaviors defined by energy management strategies as a reaction to energy condition changes. Details of the energy message handling framework is discussed in Section 2.3.

2.2 Peripheral Support: Virtual Device

Although GEM5 provides powerful simulation ability to estimate the functionalities of processor and memories, supports are still vacant for various peripherals, such as sensors and radio transmitter, which play indispensable roles in energy harvesting systems. To support the peripherals, this subsection first gives a general model of a peripheral and then introduces the proposed hardware component, virtual device.

Peripheral Modeling Figure 2 (a) shows the structure model a typical digital peripheral. The model contains a digital part to realize data storage and control logics, and an analog part to realize specific functionality. The digital part contains an internal register file to store the command, execution status and data. The processor can access a peripheral by writing commands to the *cmd_reg* and read from the *data_reg*. When a new command arrives, the changing bits in the *cmd_reg* will trigger operations, such as initialization, sensing and radio transmission.

Figure 2 (b) shows a typical peripheral working flow. A peripheral needs to be initialized to get ready before executing other tasks. When a task (collecting a data) is completed, the peripheral will trigger an interrupt to inform the processor for subsequence operations.

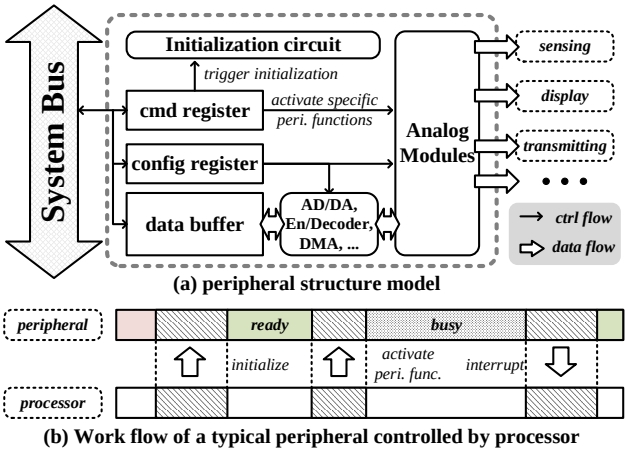

Figure 2: The structure model (a) and a typical working flow (b) of common digital peripherals.

Virtual Device Object EBeSS proposes a configurable virtual device object defining the peripheral behaviors and the interconnections between peripherals and the processor, as shown in Figure 3 (a). Virtual device is a memory-like module that the processor can make a request by accessing the address of virtual device. When the processor accesses a registered virtual address, the system will forward this access to the physical address of the related virtual device. The address of virtual devices consists of a control byte and a maximum 511B internal memory space bytes as data bytes, as shown in Figure 3. In the control byte, four bits are used to control the executing status of peripherals. *addr1* is a configurable *init* bit

to trigger the initialization operation. *addr2* is a read-only *ready* bit representing whether a device is initialized. *addr5* is a read-only *complete* bit and will be set and trigger an interrupt to the processor, when a virtual device operation is finished. *addr6* is a read-only *busy* bit representing whether the device is during execution and is not accessible if the bit is set. *addr7* is a configurable bit set by CPU to make a request to a virtual device.

EBeSS requires users to modify peripheral-related programs to suit the design of virtual device. Figure 3 (b) shows a program of temperature sensing task. Before all, the sensor has to declare its virtual space range and the address of its *cmd_reg*. In energy limited scenarios, energy harvesting system has no operating system to support automatic allocate address spaces for peripheral. Therefore, EBeSS allows virtual devices to register virtual addresses. Then, the initialization command is load to the *cmd_reg* to prepare the sensor. After the sensor state is ready, the system activates the sensor to start a sensing tasks. When the sensing task is completed, the processor checks the *complete* state bit and fetch the collected data from *data_reg*.

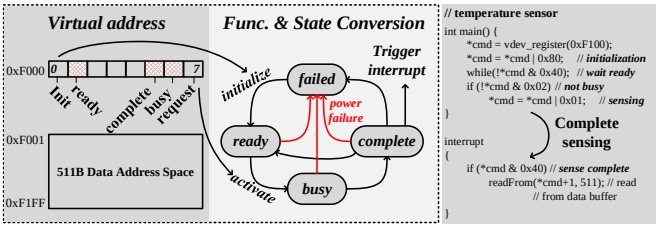

Figure 3: Structure of virtual device object and an example program of a temperature sensor.

2.3 Energy Message Handling Framework

Energy message handling framework (EMHF) is a message-based event handling framework to support the energy managing strategies in a self-powered system. The framework consists of three components: the informer, the connector and the reactor, as shown in Figure 4. With these components, EMHF realize the functionality of energy harvesting, monitoring and all the detailed energy-related behaviors of each hardware component according to energy managing strategies.

Informer: Energy Management Module. In the energy message handling framework, an energy management module (EMM) is developed to monitor energy changes and broadcast state conversion messages according to a user-defined energy managing strategy. EMM consists of a energy storage, a harvester, a consumer and a state machine, as shown in Figure 5. During each time tick, the energy harvester collects the income energy from an energy profile and consumer will collect energy consumptions from all the other hardware modules. These energy changes are committed to the energy storage (capacitors). The state machine is configurable to realize and explore the optimal design of the user-defined energy managing strategies. When the supply voltage in the storage reaches certain thresholds, EMM determines state conversion messages according to the state machine and broadcast to the entire system.

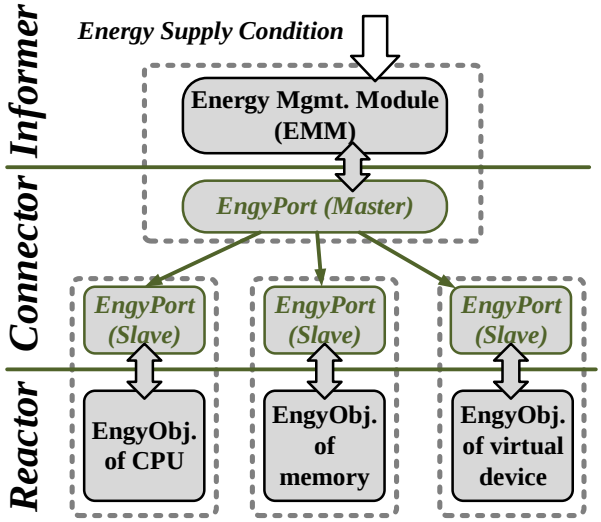

Figure 4: Structure of the energy message handling framework. The framework contains three components, the informer, the connector and the reactor.

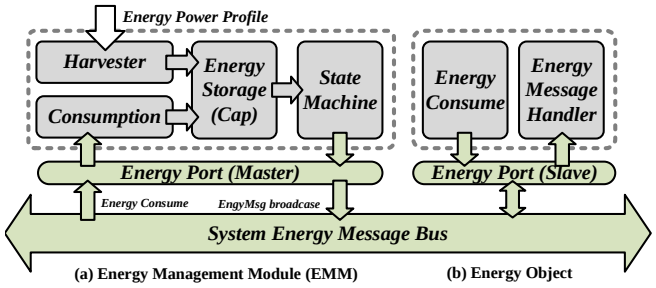

Figure 5: The structure of EMM and energy object.

Reactor: Energy Object. Except for EMM, all the other hardware modules are defined as reactors, including the processor, memory and virtual devices. In GEM5, the native simulation object for these module supports only the functionary behaviors such as computing and memory accessing operations. To support the energy-related behaviors, EBeSS extend an energy object for each hardware module. An energy object contain an energy message handler, an consumer and is connected to in EMHF framework via energy ports (introduced later), as shown in Figure 4 (b). The energy message handler is a programmable module to realize the detailed reactions defined in an energy managing strategy. According to developers' needs, the handler will adjust the run-time states and parameters, such as reaction delay, consumption, frequency and register data, to reply the upcoming energy messages. The consumer is used to manage all the run-time energy consumptions, including execution consumption and leakages. Every tick or after a message reaction, the consumer should reply its energy consumption in a specific energy consuming message format to EMM to commit the energy costs.

In future, EBeSS will open more authorities that the reactor replying more kinds of energy messages under the premise of system robustness.

Connector: Energy Ports. Energy Ports are interfaces that construct connections and exchange energy messages within EMM and energy objects. There are two types of energy ports: master port and slave port. Master ports are connected to multiple slave ports and maintain a list of its slaves during simulation. In EBeSS, master port is adopted in EMM module as global energy message informer to actively broadcast energy messages in the entire system. Slave ports are can only track its unique master. In EBeSS, all the reactors (hardware modules) are connected to slave ports which is used to receive upcoming energy message and reply consumptions. Note that, master ports can only connect to slave ports, and vice versa. All the connections are established by users determining master for each slave port in the configuration profile. Currently, only one master port is alternative in EBeSS, hence, all the slave ports have to be connected to EMM.

3 SIMULATOR VALIDATION

To validate the correctness and accuracy of EBeSS, we compare the execution parameters of EBeSS with a real NVP-based system prototype [11] as shown in Figure 6. A bridge monitoring application and several independent tasks are executed to compare the performance and energy parameters.

3.1 System configurations

The configuration of the system prototype is listed in Table **??**. This prototype is an solar-powered system, which contains a nonvolatile processor (NVP) [5, 9], a nonvolatile radio transmitter (NVRF) [11] and several volatile sensors.

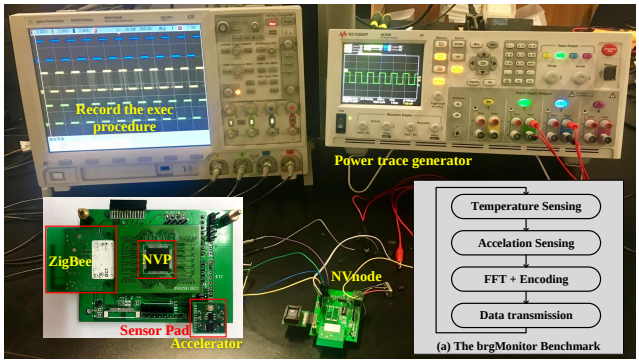

Figure 6: The hardware prototype used to validate EBeSS simulator. A real bridge monitoring benchmark is utilized to validate the simulator.

NVP enables a two-threshold backup/restore energy managing strategy (*2-thr*) [3, 9], where a lower threshold V_{low} and a higher threshold V_{high} are defined. When the supply voltage falls below V_{low}, the processor immediately backup the execution status to NVM with the help of a control circuit. When the supply voltage raises above V_{high}, the processor restarts and restores all the states and keep on progress.

NVRF is a nonvolatile interface that can automatically recover the radio transmitter and retransmit the uncompleted packet transmission after outages. Inheriting the two voltage thresholds in *2-thr*, NVRF can backup the configurations of the radio transmitter when the voltage is lower than V_{low}, and will reinitialize the transmitter states, resume the configurations and restart the uncompleted transmission.

In addition, the prototype also contains a thermometer and an accelerometer. The capacitor is $15uF$. All the above hardware parameters are listed in Table **??**. The power profiles are solar power traces from MIDC database of NREL Solar Radiation Research Laboratory [7].

3.2 Application Validation

We compare the simulation result with the hardware execution result using a real IoT application, bridge monitoring application (*brgMonitor*). The work flow of *brgMonitor* is shown in Figure 6. The device executes a 'sense-process-transmit' cycle to collect bridge health related parameters. The execution time and energy consumption comparison is shown in Table **??**.

Furthermore, we also evaluate the correctness of EBeSS with multiple small tasks in Table **??**. The maximum error is limited within XXX.

4 DESIGN SPACE EXPLORATION ON EBESS

EBeSS is an energy-aware simulator supporting peripheral and energy managing strategy configurations for self-powered system. With EBeSS, developers can optimize the hardware parameters of self-powered system (such as the capacitor size), and select appropriate energy managing strategy to achieve the maximum task completion performance. This section explores the design space of capacitor and energy managing strategy selection under different power traces and different peripheral numbers.

4.1 Capacitor Design Space Considering Peripherals

In this part, we explore the capacitor design space for self-powered system with different number of peripherals. Based on the *brgMonitor* application, we scale the peripheral number in the application from zero to five, and simulate the completed task number within 10 minutes. Figure 7 shows the design space of capacitor size in self-powered system. From the figure, we observe a design edge for a self-powered system with different occupancy of peripherals to optimize the task completion rate.

Figure 7: The task completion curve of system using multiple peripherals and different capacitor sizes.

4.2 Explore Energy Managing Strategy Selection

With an optimal hardware platform, this part explores the energy managing strategy selection with different energy supply conditions. We scans the combination of different power traces and capacitors, and explores the suitability of different energy managing strategies.

Three popular existing energy managing strategies in the energy harvesting system are analyzed, that is, backup-restore [6], state-retention [11], and DVFS [2].

- **State-retention**
  State retention is a conventional energy managing strategy that all the peripherals enter an ultra-low cost sleep mode when the power supply is not enough. During the sleeping stage, the system suffers a leakage power consumption to realize hardware states retention in the volatile memories.
- **DVFS**
  DVFS is an energy managing strategy used to dynamically adjust the system performance according to energy supply condition. DVFS enables the system to execute at lower speed with lower energy consumption when the energy supply is limited, and execute at higher speed when the energy supply is sufficient.
- **Backup-restore**
  Backup-restore is a zero-leakage energy managing strategy that can bridge across power failure conditions by backing up and restoring the volatile execution states into nonvolatile memories. By moving the states into nonvolatile memory, the system can be safely shutdown to avoid leakage power during the outages and is most reliable in energy harvesting systems.

From the definition of the three energy managing strategies, it is obvious that, the performance of these strategies is affected by the system energy supply, which is determined the power trace and the system capacitor size. Therefore, we explore the search space of the combination of power trace and capacitor size.

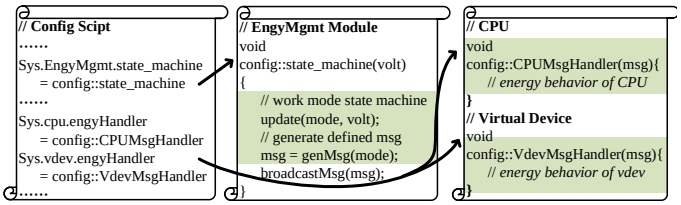

Figure 8: The task completion curve of system using multiple peripherals and different capacitor sizes.

We configure the energy managing strategy by editing the state machine in EMM to generate energy message, and the energy message handler in CPU and virtual device source code, as shown in Figure 8. Figure 9 (a-c) shows the performance hotpot graph when executing the *brgMonitor* benchmark using each of these strategies. Comparing the performance hotpot graph of these energy managing strategies, Figure 9 (d) shows the distribution of the optimal energy managing strategy selection. We can conclude that ...

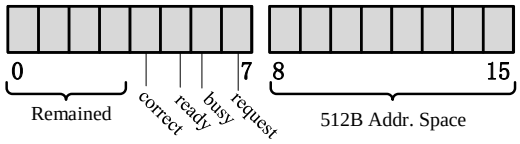

Figure 9: The task completion curve of system using multiple peripherals and different capacitor sizes.

4.3 Exploration Conclusion

In conclusion, experiment explores the affect of the power supply density and the application types. Results show that, DVFS has the ability to take better usage of the too low and the too high energy supply. On the other side, ModeCvrt and DVFS have the advantages in adjusting the energy usage of CPU and peripherals, respectively. When the computing operations is dominating, DVFS has better performance to take fully usage of power supply. When the I/O operations are dominating, ModeCvrt is the better choice to avoid the leakages of the idle peripherals.

5 CONCLUSION

Target on energy behavior design exploration, this paper proposes a NVP based energy harvesting system simulator, EBeSS, which supports flexible energy behavior configuration of both NVP and peripherals. With the help of EBeSS, we explores the performance and energy efficiency of three existing energy behaviors with different power profiles and application types. The experiments shows the insights that both DVFS and ModeCvrt schemes have advantages in different power supply and application conditions. Furthermore, EBeSS shows the potential to efficiently and correctly optimize the design parameter settings and validate the data consistency and correctness of the energy management strategies in the transiently powered energy harvesting systems.

REFERENCES

[1] Alexei Colin, Graham Harvey, Brandon Lucia, and Alanson P. Sample. 2016. An Energy-interference-free Hardware-Software Debugger for Intermittent Energy-harvesting Systems. *ASPLOS 2016, Acm Sigops Operating Systems Review* 50, 2 (2016), 577–589.

[2] Benjamin J Fletcher, Domenico Balsamo, and Geoff V Merrett. 2017. Power neutral performance scaling for energy harvesting MP-SoCs. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1516–1521.

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

[4] Zewei Li, Yongpan Liu, Daming Zhang, Chun Jason Xue, Zhangyuan Wang, Xin Shi, Wenyu Sun, Jiwu Shu, and Huazhong Yang. 2016. HW/SW co-design of nonvolatile IO system in energy harvesting sensor nodes for optimal data acquisition. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 154.

[5] Yongpan Liu, Zewei Li, Hehe Li, Yiqun Wang, Xueqing Li, Kaisheng Ma, Shuangchen Li, Meng Fan Chang, Sampson John, and Yuan Xie. 2015. Ambient energy harvesting nonvolatile processors: From circuit to system. In *Design Automation Conference*. 150.

[6] 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 *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. 526 – 537.

[7] Measurement and Instruction Data Center (MIDC). [n. d.]. http://www.nrel.gov/midc/srrl_bms/. ([n. d.]).

[8] Fang Su, Yongpan Liu, Yiqun Wang, and Huazhong Yang. 2016. A Ferroelectric Nonvolatile Processor with 46 μs System-Level Wake-up Time and 14 μs Sleep Time for Energy Harvesting Applications. *IEEE Transactions on Circuits & Systems I Regular Papers* PP, 99 (2016), 1–12.

[9] 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 *2012 Proceedings of ESSCIRC (ESSCIRC)*. IEEE, 149–152.

[10] Zhibo Wang, Yongpan Liu, Albert Lee, Fang Su, Chieh-Pu Lo, Zhe Yuan, Jingyang Li, Chien-chen Lin, Wei-Hao Chen, Hsiao-Yun Chiu, Wei-En Lin, Ya-Chin King, Chrong-Jung Lin, Pedram Amiri, Kang-Lung Wang, Meng-Fan Chang, and Huazhong Yang. 2017. A 65-nm ReRAM-Enabled Nonvolatile Processor With Time-Space Domain Adaption and Self-Write-Termination Achieving ¿4× Faster Clock Frequency and ¿6× Higher Restore Speed. In *IEEE Journal of Solid State Circuits (JSSC)*. IEEE.

[11] Zhibo Wang, Fang Su, Zeiwei Li, Xueqing Li, Ryuji Yoshimura, Takashi Naiki, Takashi Tsuwa, Takahiko Saito, Zhongjun Wang, Koji Taniuchi, Meng-Fan Chang, Huazhong Yang, and Yongpan Liu. 2017. A 130nm FeRAM-based parallel recovery nonvolatile SOC for normally-OFF operations with 3.9× faster running

speed and 11× higher energy efficiency using fast power-on detection and non-volatile radio controller. In *Symposia on VLSI Technology and Circuits*. IEEE, C336–C337.