HomeRun: HW/SW Co-Design for Program Atomicity on Self-Powered Intermittent Systems

Chih-Kai Kang[1,3], Chun-Han Lin[2], Pi-Cheng Hsiu[1], Ming-Syan Chen[1,3]
[1]Research Center for Information Technology Innovation, Academia Sinica, Taiwan
[2]Department of Computer Science and Information Engineering, National Taiwan Normal University, Taiwan
[3]Graduate Institute of Electrical Engineering, National Taiwan University, Taiwan
Email: akaikang@citi.sinica.edu.tw, chlin@csie.ntnu.edu.tw, pchsiu@citi.sinica.edu.tw, mschen@ntu.edu.tw

ABSTRACT

Self-powered intermittent systems featuring nonvolatile processors (NVPs) allow for accumulative execution in unstable power environments. However, frequent power failures may cause incorrect NVP execution results due to invalid data generated intermittently. This paper presents a HW/SW co-design, called HomeRun, to guarantee atomicity by ensuring that an uninterruptible program section can be run through at one execution. We design a HW module to ensure that a power pulse is sufficient for an atomic section, and develop a SW mechanism for programmers to protect atomic sections. The proposed design is validated through the development of a prototype pattern locking system. Experimental results demonstrate that the proposed design can completely guarantee atomicity and significantly improve the energy utilization of self-powered intermittent systems.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; • **Software and its engineering** → *Correctness*;

KEYWORDS

Program atomicity, energy utilization, nonvolatile processors, intermittent systems

1 INTRODUCTION

The popularity of internet of things (IoT) applications and wearable devices is increasing exponentially, and the effective lifetime of such devices is a crucial consideration for quality of user experience. Most devices are powered by batteries which may incur expensive maintenance costs and serious environmental pollution. To avoid these problems, these devices use energy systems based on energy harvesting units, allowing them to operate intermittently for long periods without the use of external power sources. The power harvested from an ambient source is fundamentally small and unstable, particularly in self-powered intermittent systems, resulting in frequent power failures. *Nonvolatile processors* (NVPs) are seen as a promising alternative for use in self-powered intermittent systems. Compared to typical micro controller units (MCUs), NVPs feature characteristics including zero standby power and resilience to power failures, making them appropriate for small and unstable energy systems [5, 19]. However, the new NVP-based self-powered intermittent systems raise new problems that do not exist in traditional MCU-based battery-powered systems [7].

Previous research on NVP-based systems can be categorized into three classes, namely *hardware circuits*, *system architecture*, and *system software*, according to design levels [5, 26]. Many researchers have studied the hardware circuit level (e.g., flip-flop [13, 17] and controller [6]) focusing on improving the efficiency of the backup circuits for NVPs. Much work has also been done on the system architecture level (e.g., processor architecture [2, 9, 19, 22] and power supply architecture [1, 10, 14]), aiming to improve backup operations efficiency. These efforts have led to the development of critical components for NVP-based self-powered intermittent systems.

Recently, much attention has been devoted to the system software level (e.g., scheduler [23, 24] and various design tools [11, 18, 21, 25]), which helps increase program developers' productivity. Some checkpointing [20] and versioning techniques [8] have been proposed to avoid *data inconsistency* between different system snapshots in the memory hierarchy. To consider *program atomicity* where some code sections cannot be executed intermittently, a safe execution model has been proposed to roll back partially executed code to avoid repeated code execution [8]. However, program progress cannot be guaranteed by repeatedly re-executing an atomic section until an execution attempt eventually completes without interruption, and how to ensure that an atomic section can be run through remains a challenge.

In this paper, we first show an example in which the execution result of a program may be invalid if a code section is executed intermittently. The functionality of the program is maintained correctly by the NVP, but the data becomes invalid due to unexpected time difference caused by a power failure. In the example, we show that a combined image is invalid if a single image consists of data captured at a different time. Therefore, in NVP-based self-powered intermittent systems, a power-efficient mechanism that guarantees no intermittent execution in some program sections is necessary to ensure program atomicity.

To address the issue, we present a hardware and software co-design, called HomeRun, which allows developers to avoid invalid data and improve energy utilization. First, we present an atomic-aware software library to ensure program atomicity. We then present a general structure of a hardware circuit to guarantee that a self-powered intermittent system has sufficient energy to execute an atomic section. By using HomeRun, we implement a pattern locking system with self-powered photodiodes to control the lock state based on user input without relying on external power sources. We conduct experiments on the pattern locking system to evaluate the efficacy of the proposed

ISLPED '18, July 23–25, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5704-3/18/07…$15.00
https://doi.org/10.1145/3218603.3218633

design. The results show that HomeRun can significantly improve energy utilization and increase the program completeness count.

The remainder of this paper is organized as follows. Section 2 provides background information and describes our motivation. In Section 3, we present the hardware and software co-design of HomeRun for NVP-based self-powered intermittent systems. Section 4 describes a pattern locking system implemented with HomeRun. The experimental results are reported in Section 5. Section 6 provides some concluding remarks.

2 BACKGROUND AND MOTIVATION

This section presents a typical self-powered intermittent system equipped with an emerging NVP. We then show that supporting atomic operations on such systems are necessary to obtain valid program results.

2.1 NVP-Based Self-Powered Intermittent Systems

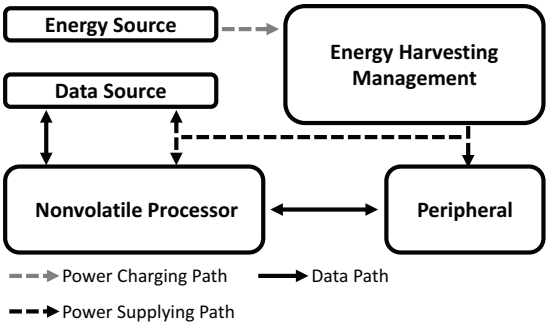


Figure 1: An NVP-based self-powered intermittent system

A typical NVP-based self-powered intermittent system consists of an energy source, an energy harvesting management (EHM) unit, an NVP, a data source, and peripherals, as shown in Figure 1. According to the types of ambient sources, the energy source may be photovoltaic, piezoelectric, or other harvesting devices [5, 12]. The energy source harvests energy from the corresponding ambient source, and then transmits the harvested energy to the EHM unit through a power charging path. When the harvested energy is sufficient, the EHM unit will supply power to the system, and store redundant energy to its capacitors for spare usage as needed. When the NVP is woken up, it uses the previous backup to restore the program state, and then continues executing programs, collecting data from the data source, operating peripherals, and backing up the program state. The NVP may support a periodic or on-demand backup mechanism to handle the power failure problem [16]. When a power failure occurs, the NVP suspends until sufficient energy has been harvested and resumes instantly from where it left off after the power is restored. Therefore, an NVP-based self-powered system with an energy harvesting unit can intermittently operate over long periods without any external power source.

2.2 Atomic Sections

A program section that must be executed as one uninterruptible unit is called an atomic section. Figure 2 shows an example where the execution result of an atomic section may be invalid if it is executed intermittently. In the example, the NVP-based self-powered

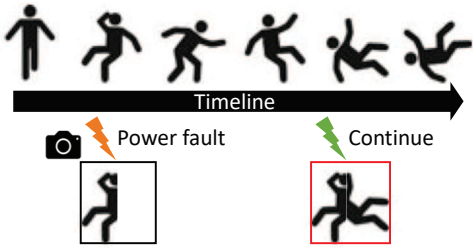


Figure 2: An example of invalid data

intermittent system executes a program which contains one task to capture images with an array of image sensors. The capturing task sequentially scans every image sensor and combines all sensor output into one image after completing the scanning task. If power failure occurs in the middle of the scanning task, the NVP will backup the current program state, to be restored and continued at the next power pulse. Although the NVP can correctly maintain the program functionality, the data may be invalid due to intermittent collection caused by a power failure, because the scene may change while the system is off. The combined image is clearly invalid because the program developer does not anticipate a single image comprising data captured at different times. However, programs may contain some code sections that cannot be interrupted. Therefore, in NVP-based self-powered intermittent systems, a power-efficient mechanism that guarantees no intermittent execution in some program sections is needed for program developers to designate and protect atomic sections.

3 ATOMIC-AWARE HW/SW CO-DESIGN

This section presents an atomic-aware design with efficient energy utilization on NVP-based self-powered intermittent systems. In Section 3.1, we give a design overview which contains functions based on an atomic-aware software library and an energy guaranteed hardware circuit. We then detail the software library and the hardware circuit in Sections 3.2 and 3.3, respectively.

3.1 Design Overview

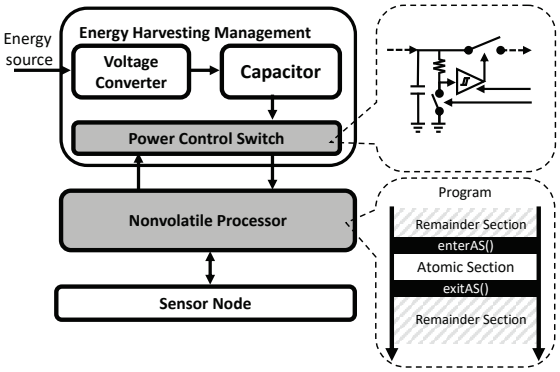


Figure 3: An overview of our atomic-aware HW/SW co-design

The proposed atomic-aware HW/SW co-design, HomeRun, consists of an atomic-aware library and a hardware circuit, as shown in Figure 3. For program developers, handling the invalid-data problems caused by intermittent executions is difficult because of the potential for

unexpected power failures, and it may reduce developers' productivity. We propose a software library that can be integrated into the system software on the NVP to address the issue. The library provides convenient functions for program developers to protect atomic sections. Developers only have to designate an atomic section by adding an entry function and an exit function respectively before and after the atomic section. However, if the energy provided from the EHM unit is insufficient to finish an atomic section, the program will be jammed in the atomic section. To meet the energy requirements of atomic sections, we then propose a hardware circuit that can be integrated in the EHM unit to ensure program progress by providing the minimum energy sufficient to complete the sections. To further avoid possible energy wastage in the first execution of each atomic section, HomeRun automatically stops program execution if the remaining energy is insufficient. Consequently, HomeRun efficiently guarantees program atomicity by ensuring that an atomic section can be run through at one execution.

3.2 Software Library

Power failures may result in partial task execution, resulting in invalid data. To maximize the energy utilization and program correctness, an *atomic-aware library* is proposed to guarantee that atomic sections which contain continuous code must be executed as one uninterruptible unit. Prior to entering an atomic section, a program must ask the system to enter the section in an *entry section*. The atomic section then is followed by an *exit section*. The program's remaining code is the remainder section. Assume that the energy of a power pulse from the EHM unit is sufficient for an atomic section. The proposed atomic-aware library contains two functions to allow developers to easily construct an atomic section in a program.

The function designed for the entry section is defined in pseudo code as follows.

```
1 enterAS(V_d, *stopAutoBackup()){
2   stopAutoBackup(); // save the backup energy
3   backup(); // save the program progress
4   // save the remaining energy
5   if(current energy ≤ V_d)
6     switch the power supplying path off;
7 }
```

The NVP automatically triggers backup and restore operations, so a program is able to continue from its last backup position in the previous power pulse. Continuing a program in the middle of an atomic section is a potential risk, so a backup operation executed in an atomic section shall not be used to continue a program and is viewed as an energy waste operation. Therefore, an essential task of the entry section is using the function, `stopAutoBackup()`, provided by developers, to disable relevant interrupt signals[1] to prevent the NVP from *automatically* triggering a backup operation. However, a subsequent power pulse may proceed from the same position of the previous one, and there is a high probability of a power failure occurring at a similar position, so the program will fail to progress and the energy of the following power pulse is wasted as well. Therefore, the entry section executes a backup operation to allow the program to continue from here if intermitted, thereby preserving program progress to increase the probability of finishing an atomic section in the following power pulse. Furthermore, the remaining energy of the current power pulse may be insufficient to complete the atomic section. If the program begins the atomic section with insufficient energy, the NVP will stop in middle of the atomic section and restart at Line 5 in the entry section, so the energy used in the previous execution is wasted. Therefore, the entry section checks the amount of remaining energy with a *developer-defined threshold* V_d, and only continues execution when the amount is higher than V_d. Otherwise, the entry section switches off the power supply path to preserve the remaining energy, and switches the power supply path on until sufficient energy is available.

The function designed for the exit section is defined in pseudo code as follows.

```
exitAS(*startAutoBackup()){
  startAutoBackup();
}
```

After finishing an atomic section, the exit section must use the function, `startAutoBackup()`, provided by developers, to enable relevant interrupt signals to turn on the automatic backup mechanism. The NVP will then backup automatically if necessary, so a backup operation is unnecessary in the exit section. This software library is currently designed for single-thread applications, and it can be extended to multi-thread applications by wrapping all `enterAS()` and `exitAS()` function pairs in a mutex lock to avoid a race condition [15].

3.3 Hardware Circuit

A program will be jammed in an entry section if the energy of the subsequent power pulse is insufficient to finish an atomic section, so a design which is purely based on the software library cannot ensure program progress. Therefore, the characteristics of the EHM unit shall be designed properly based on the characteristics of the hardware and software of the NVP-based self-powered intermittent system. Figure 4 shows the general structure of the energy storage and the power control switch to accumulate harvested energy and generate power pulses to meet the energy requirements of atomic sections. The EHM unit normally uses capacitors to store harvested energy, and the energy in a capacitor with capacitance C and current voltage V_c is equal to $\frac{1}{2}CV_c{}^2$. In the hardware circuit, as the switch we use a Schmitt trigger, which only changes its switch state when the voltage crosses one of the two predefined thresholds. The switch will toggle the power supply path on when the capacitor voltage exceeds the operating threshold V_o and off when it is lower than the cutting threshold V_h. Generally, V_o and V_h are specified by the hardware vendor. Therefore, the energy of a power pulse from the power control switch is equal to $\frac{1}{2}C(V_o^2 - V_h^2)$. In addition, the NVP can send a signal to switch the power supply path off if the entry section finds that the amount of remaining energy is insufficient to complete an atomic section. The remaining energy is equal to $\frac{1}{2}C(V_c^2 - V_h^2)$, which is dependent on the current capacitor voltage V_c, so we can determine whether the remaining energy is sufficient to pass an atomic section by checking the current capacitor voltage.

The energy of a power pulse has to be equal to or greater than the energy required for finishing any atomic section, because all atomic sections must be executed without interruption. According to the atomic-aware library, the minimum energy requirement of an atomic

[1]Typically, an NVP automatically triggers a backup operation via an interrupt from a periodic timer or a low-voltage warning signal.

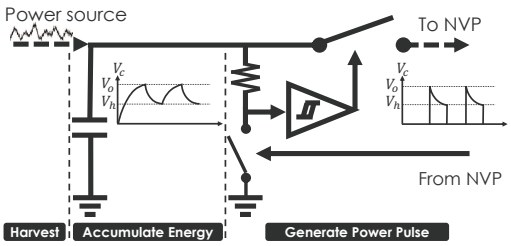


Figure 4: An illustration of energy storage and control switch

section happens when the entire energy of a power pulse is used to complete the whole atomic section, so its minimum energy is equal to the sum of the energy consumption of a restore operation RE, the atomic section, and a backup operation BK. The energy used to start and stop the auto backup mechanisms is usually negligible. Figure 5 shows the power distribution of a typical program. A program can complete its functions correctly if the energy of a power pulse satisfies the minimum energy requirements of all atomic sections. The energy E_i required for the i_{th} atomic section is equal to $E(RE) + E(AS_i) + E(BK)$, where $E()$ and AS_i respectively represent the energy function and the atomic section. Accordingly, the minimum energy requirement E_{min} with n atomic sections can be expressed as $E_{min} = \max_{i=1}^{n} E_i$, where the energy requirement E_i and the corresponding voltage threshold $V_{d,i}$ can be derived based on the methods presented in [3, 4].

Based on $V_{d,i}$, the system could determine whether the remaining energy is sufficient to complete AS_i. Note that if $V_{d,i}$ is underestimated so that the system runs the atomic section with insufficient energy, the section will be run through in the next fully-charged power pulse. Therefore, the proposed hardware circuit can satisfy the minimum energy requirement by adjusting the three values, C, V_o, and V_h. Specifically, the energy of a power pulse, $\frac{1}{2}C(V_o^2 - V_h^2)$, must be equal to or greater than E_{min} to meet the minimum energy requirement.

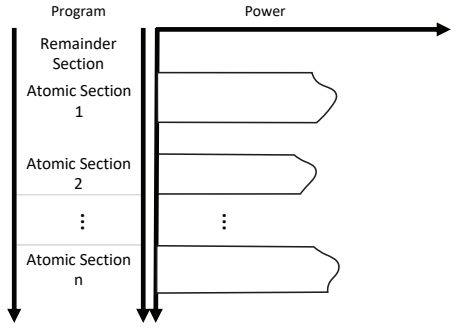


Figure 5: Power distribution of a typical program

4 A PATTERN LOCKING SYSTEM

As a pedagogical example, we develop a pattern locking system based on the proposed HW/SW co-design, HomeRun. Figure 6(a) shows the pattern locking platform composed of a self-powered intermittent system and an activator. Two operation modes, HomeRun and Native, are implemented in the pattern locker, and the operation mode of the activator can be toggled with a button. The self-power system collects data from the sensors intermittently (depending on the power source that may vary over time), and then sends a signal to the activator according to the results of the pattern matching process. Some system information, such as the lock status or program completeness count, will be computed and displayed on the activator's LCD.

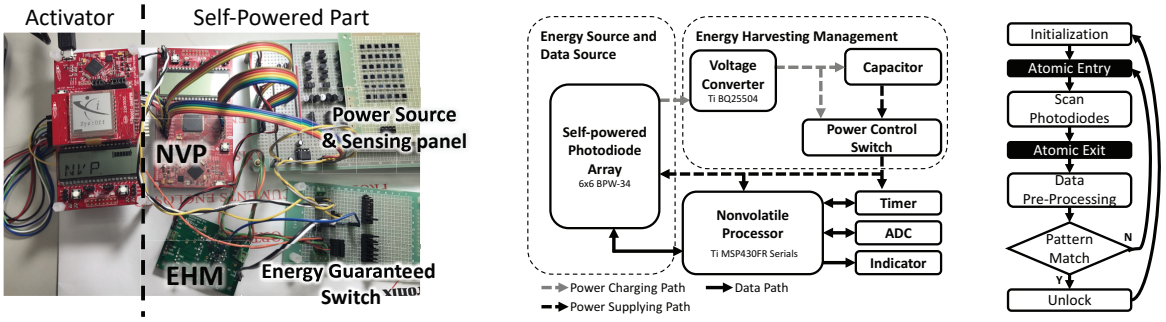
The pattern locking system controls its locking state based on the collected data without any external power, as shown in Figure 6(b). Its self-powered sensors are implemented as an array of 6×6 BPW34 photodiodes to harvest light energy and collect light data. The EHM unit consists of a BQ25504 low-power boost converter, capacitors, and a switch to store the harvested energy and toggle the power supply path. The converter provides power management capability to operate as the required Schmitt trigger. To execute the pattern matching programs, the NVP is implemented as a MSP430FR series micro controller which integrates a nonvolatile Ferroelectric RAM. Peripherals consist of a timer, an analog-to-digital converter (ADC), and 83 general-purpose input/output (GPIO) pins to trigger the automatic backup mechanism, collect lighting data, and show the locking state.

We implement a periodic backup mechanism with a `backup()` function and a periodic timer, because the MSP430FR series MCU has no automatic backup mechanism[2]. The function sequentially backs up the system stack and the stack pointer to the FRAM. Note that the registers like the program counter are not backed up, because the register data, including the returned address which represents the program counter in the pattern locking system, have been pushed into the system stack when the `backup()` function is called. After implementing the function, we implement an auto backup mechanism by setting a periodic timer to trigger the function periodically. Therefore, the functions to start and stop the auto backup mechanism are respectively implemented by enabling and disabling the periodic timer. The restore operation is implemented as a `restore()` function whose main steps are symmetrical to the `backup()` one. The restore function is added in a boot hook function which is invoked for system pre-initialization when the MCU is activated each time.

We use the software library in HomeRun to implement the atomic-aware pattern matching program, as shown in Figure 6(c). By default, the program keeps the system in the locked state, and then unlock the system when the collected data matches a default pattern. The program repeatedly collects data from the photodiodes. Whenever all the photodiodes are scanned, the collected raw data is first normalized to a standard format, and then compared against every default pattern. If a match is found, the system is temporarily unlocked, then relocks immediately after the system cleans all its status and returns to the initial state, and the process repeats. In contrast, if a mismatch is found, the system scans the next pattern input while keeping locked. The combined raw data collected during different time periods may be invalid and cannot be used as an input pattern. Therefore, the pattern matching process requires uninterrupted scanning of the 6×6 photodiodes, and the scanned photodiode block should be protected as an atomic section. Therefore, the `enterAS()` and `exitAS()` functions in HomeRun are respectively added before the first photodiode and after the last photodiode are scanned.

We use the hardware circuit in HomeRun to design the EHM unit of the pattern locking system. To prevent the supplied voltage from exceeding the NVP's operating voltages due to the response delay of the converter, the operating threshold V_o is set to be slightly lower than the NVP's maximum operating voltage, and the harvesting threshold V_h is set slightly higher than the NVP's minimum operating voltage. We then

[2]The implementation of backup and restore mechanisms is unnecessary on system testbeds equipped with NVPs, e.g., the NVP chip developed in [19].



(a) System platform (b) Hardware architecture (c) Software flowchart

Figure 6: A pattern locking system

adjust the capacitor's capacitance C to satisfy the energy requirement of the atomic-aware pattern matching program.

5 PERFORMANCE EVALUATION

5.1 Experimental Setup

To evaluate the performance and better understand the properties of our atomic-aware design, we conducted a series of experiments on the pattern locking system presented in the previous section. The related system specifications are detailed in Table 1. The proposed design, HomeRun, aims to improve energy utilization and ensure program correctness. Therefore, its performance is evaluated in terms of the number of times the programs are completed, and the correctness ratio of the collected data. If a power failure causes an interruption in middle of an atomic section, the collected data is invalid, and the correctness ratio is the ratio of the number of valid data to the total number of data collected. We compare HomeRun with the native system, denoted as Native, under strong and weak power sources. Native is an NVP-based system having the same hardware architecture and automatic backup/resotre mechanisms, but it does not use the proposed atomic-aware software design. We use the Keithley 2280S DC power supply to generate reproducible power traces.

Table 1: System specifications

Item descriptions	Values
Average power consumption	1.6 mW
Task execution time	50 ms
Execution time of atomic section	22 ms
Operating threshold (V_o)	2.8 V
Cutting threshold (V_h)	2.2 V
Strong power source	1 mW (1 V, 1 mA)
Weak power source	500 uW (0.5 V, 1 mA)

We conducted two sets of experiments to evaluate HomeRun from different perspectives. First, we investigated the impact of voltage threshold V_d in the range 2.0 V to 2.5 V with a 300 uF capacitor. This set of experiments provides some insights for developers to understand the effects of a parameter in the entry section. In the second set of experiments, we investigated the impact of various capacitance levels of energy storage ranging from 100 uF to 900 uF. This set of experiments compares HomeRun and Native under different conditions. All of the experiments were tested under strong and weak power source conditions. Note that the system runs the program intermittently because the power from the power source is less than total system power consumption.

5.2 Impact of Different Voltage Thresholds

Figure 7 shows the impact of different voltage thresholds V_d on the program completeness count. The number of times the programs are completed is maximized for both power conditions when the threshold equals 2.2 V. When V_d is too small, the remaining energy after passing the entry section may be insufficient to complete the atomic section. As a result, the system needs to rerun the atomic section once (and only once), requiring one additional retry to pass through the section in the next fully-charged power pulse. In contrast, when the V_d is too large, the energy between the operating voltage and the threshold may be too small to complete a restore operation. Therefore, when system wakes up, the remaining energy after a restore operation may not be sufficient to pass the check in an entry section, and the entry section will switch off the power supply path. Based on these observations, the default setting of the voltage threshold is set at 2.2 V.

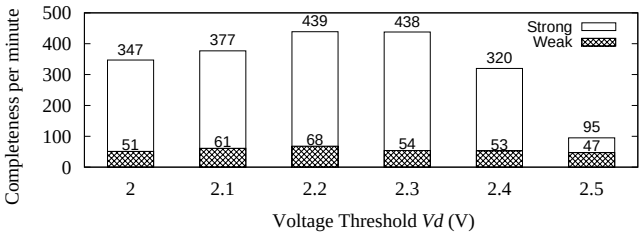


Figure 7: Impact of different voltage thresholds V_d

5.3 Impact of Various Capacitance Levels

Table 2 shows the impact of various capacitance levels on program completeness per minute. In all cases, the program completion rate increases with capacitance because a large capacitance level can provide a large amount of energy, and thus can provide much program progress. However, program completeness decreases after the capacitance exceeds 500 uF because larger capacitance requires greater charging time. In most cases, the program completeness count in HomeRun is higher than that in Native because HomeRun stops the auto backup mechanism in the atomic sections, and this design saves energy from useless backup operations in middle of atomic sections. Compared with Native, on average HomeRun respectively improves the program completeness count by 39% and 20% for strong and weak power sources.

Table 3 shows the impact of various capacitance levels on the correctness ratio of the collected data. The correctness ratio in HomeRun is always 100% because HomeRun prevents programs from stopping in the middle of atomic sections. The correctness

Table 2: Program completeness per minute

Capacitance \ Power	Strong (1mW)		Weak (500uW)	
	Native	HomeRun	Native	HomeRun
100 uF	249	332	35	35
200 uF	284	385	43	44
300 uF	305	439	49	62
400 uF	314	447	53	68
500 uF	339	489	58	78
600 uF	349	475	59	75
700 uF	311	431	60	71
800 uF	309	422	59	71
900 uF	292	416	52	66

ratio achieved by Native increases with capacitance because high capacitance provides a large amount of energy in each power pulse, and this increases the probability of completing an atomic section without interruption. However, the correctness ratio is saturated when capacitance exceeds 500 uF because a single power pulse provided by a large capacitance can complete the whole program multiple times, and each power pulse can only cause invalid data once, so the increasing slope of the correctness ratio will become saturated. In summary, both program completeness count and correctness ratio in HomeRun is higher than in Native. The correctness ratio represents total energy utilization because a program execution containing invalid data shall be considered to be incorrect and the energy used in execution is a waste. Therefore, HomeRun can provide 100% energy utilization, considerably outperforming Native. This characteristic is very useful given a weak power source and small energy storage, because less than 10% of the data collected by Native is valid. Overall, compared with Native, on average HomeRun respectively improves the program completeness count and correctness ratio by 29% and 39%.

Table 3: Correctness ratio of collected data

Capacitance \ Power	Strong (1mW)		Weak (500uW)	
	Native	HomeRun	Native	HomeRun
100 uF	47 %	100 %	7.3 %	100 %
200 uF	54.9 %	100 %	23.3 %	100 %
300 uF	68.5 %	100 %	40.8 %	100 %
400 uF	70.7 %	100 %	47.2 %	100 %
500 uF	76.1 %	100 %	62.1 %	100 %
600 uF	79.4 %	100 %	66.1 %	100 %
700 uF	80.4 %	100 %	68.3 %	100 %
800 uF	84.5 %	100 %	69.5 %	100 %
900 uF	84.9 %	100 %	71.2 %	100 %

6 CONCLUSION

We have presented HomeRun, a hardware/software co-design for atomicity on self-powered intermittent systems containing an atomic-aware software library and an energy guaranteed hardware circuit. HomeRun eases the protection of atomic sections by allowing developers to simply designate those uninterruptable code segments in their programs. Experimental results conducted on a real-world platform show that the proposed design improves energy utilization by increasing the program completeness count and ensuring the generation of valid data. The count of program completeness achieved with HomeRun is improved by up to 44%. Moreover, the ratio of valid data is always 100%, while the ratio without HomeRun is averagely 61% and decreases to only 7.3% when the capacitance is small. These lead to an average improvement of 39% in energy utilization. Future work will focus on the continued development of the atomic-aware design and developing a toolkit by which developers can further optimize energy efficiency through adjusting hardware and software characteristics.

REFERENCES

[1] V. A. Boicea. Energy storage technologies: The past and the present. *Procs. of the IEEE*, 102(11):1777–1794, 2014.

[2] 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 *Proc. of IEEE VLSI*, pages 330–335, 2014.

[3] H. Jayakumar, A. Raha, J. R. Stevens, and V. Raghunathan. Energy-aware memory mapping for hybrid fram-sram mcus in intermittently-powered iot devices. *ACM TECS*, 16(3):65:1–65:23, 2017.

[4] Z. Li, Y. Liu, D. Zhang, C. J. Xue, Z. Wang, X. Shi, W. Sun, J. Shu, and H. Yang. 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*, pages 154:1–154:6, 2016.

[5] Y. Liu, Z. Li, H. Li, Y. Wang, X. Li, K. Ma, S. Li, M.-F. Chang, S. John, Y. Xie, J. Shu, and H. Yang. Ambient Energy Harvesting Nonvolatile Processors: From Circuit to System. In *Proc. of ACM/IEEE DAC*, pages 1–6, 2015.

[6] Y. Liu, F. Suy, Z. Wangy, and H. Yang. Design exploration of inrush current aware controller for nonvolatile processor. In *Procs. of IEEE NVMSA*, pages 1–6, 2015.

[7] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel. Intermittent Computing: Challenges and Opportunities. In *Proc. of SNAPL*, pages 8:1–8:14, 2017.

[8] B. Lucia and B. Ransford. A simpler, safer programming and execution model for intermittent systems. In *Proc. of ACM PLDI*, pages 575–585, 2015.

[9] K. Ma, X. Li, S. Li, Y. Liu, J. J. Sampson, Y. Xie, and V. Narayanan. Nonvolatile processor architecture exploration for energy-harvesting applications. *IEEE Micro*, 35(5):32–40, 2015.

[10] D. Porcarelli, D. Brunelli, M. Magno, and L. Benini. A multi-harvester architecture with hybrid storage devices and smart capabilities for low power systems. In *Procs. of IEEE SPEEDAM*, pages 946–951, 2012.

[11] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on RFID-scale devices. In *Proc. of ACM ASPLOS*, pages 159–170, 2011.

[12] S. Roundy, D. Steingart, L. Frechette, P. Wright, and J. Rabaey. Power Sources for Wireless Sensor Networks. In *Proc. of IEEE EWSN*, pages 1–17, 2004.

[13] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai. Nonvolatile Magnetic Flip-Flop for Standby-Power-Free SoCs. *IEEE J. Solid-State Circuits*, (8):2244–2250, 2009.

[14] X. Sheng, C. Wang, Y. Liu, H. G. Lee, N. Chang, and H. Yang. A high-efficiency dual-channel photovoltaic power system for nonvolatile sensor nodes. In *Procs. of IEEE NVMSA*, pages 1–2, 2014.

[15] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts*. Wiley Publishing, 8th edition, 2008.

[16] F. Su, Z. Wang, J. Li, M.-F. Chang, and Y. Liu. Design of Nonvolatile Processors and Applications. In *Proc. of IEEE VLSI-SoC*, pages 1–6, 2016.

[17] J. Wang, Y. Liu, H. Yang, and H. Wang. A Compare-and-Write Ferroelectric Nonvolatile Flip-Flop for Energy-Harvesting Applications. In *Proc. of IEEE ICGCS*, pages 646–650, 2010.

[18] Y. Wang, H. Jia, Y. Liu, Q. Li, C. J. Xue, and H. Yang. Register allocation for hybrid register architecture in nonvolatile processors. In *Procs. of IEEE ISCAS*, pages 1050–1053, 2014.

[19] 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 *Procs. of IEEE ESSCIRC*, pages 149–152, 2012.

[20] M. Xie, M. Zhao, C. Pan, J. Hu, Y. Liu, and C. J. Xue. Fixing the broken time machine: Consistency-aware checkpointing for energy harvesting powered non-volatile processor. In *Proc. of ACM/IEEE DAC*, pages 1–6, 2015.

[21] Yizi Gu, Y. Liu, Y. Wang, H. Li, and H. Yang. NVPsim: A simulator for architecture explorations of nonvolatile processors. In *Proc. of ACM/IEEE ASP-DAC*, pages 147–152, 2016.

[22] W.-k. Yu, S. Rajwade, S.-E. Wang, B. Lian, G. E. Suh, and E. Kan. A non-volatile microcontroller with integrated floating-gate transistors. In *Procs. of IEEE/IFIP DSN-W*, pages 75–80, 2011.

[23] D. Zhang, S. Li, A. Li, Y. Liu, X. S. Hu, and H. Yang. Intra-task scheduling for storage-less and converter-less solar-powered nonvolatile sensor nodes. In *Procs. of IEEE ICCD*, pages 348–354, 2014.

[24] D. Zhang, Y. Liu, X. Sheng, J. Li, T. Wu, C. J. Xue, and H. Yang. Deadline-aware task scheduling for solar-powered nonvolatile sensor nodes with global energy migration. In *Procs. of ACM/IEEE DAC*, pages 1–6, 2015.

[25] M. Zhao, L. Qingan, X. Mimi, Y. Liu, J. Hu, and C. J. Xue. Software assisted non-volatile register reduction for energy harvesting based cyber-physical system. In *Procs. of ACM DATE*, pages 567–572, 2015.

[26] M. Zhao, K. Qiu, Y. Xie, J. Hu, and C. J. Xue. Redesigning Software and Systems for Non-Volatile Processors on Self-Powered Devices. In *Proc. of IEEE VLSI*, pages 1–6, 2016.