DIAC: Design Exploration of Intermittent-Aware Computing Realizing Batteryless Systems

Sepehr Tabrizchi*, Shaahin Angizi†, Arman Roohi*
*School of Computing, University of Nebraska–Lincoln, Lincoln NE, USA
†Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ, USA
aroohi@unl.edu

***Abstract*—Battery-powered IoT devices face challenges like cost, maintenance, and environmental sustainability, prompting the emergence of batteryless energy-harvesting systems that harness ambient sources. However, their intermittent behavior can disrupt program execution and cause data loss, leading to unpredictable outcomes. Despite exhaustive studies employing conventional checkpoint methods and intricate programming paradigms to address these pitfalls, this paper proposes an innovative systematic methodology, namely `DIAC`. The DIAC synthesis procedure enhances the performance and efficiency of intermittent computing systems, with a focus on maximizing forward progress and minimizing the energy overhead imposed by distinct memory arrays for backup. Then, a finite-state machine is delineated, encapsulating the core operations of an IoT node, sense, compute, transmit, and sleep states. First, we validate the robustness and functionalities of a DIAC-based design in the presence of power disruptions. DIAC is then applied to a wide range of benchmarks, including ISCAS-89, MCNS, and ITC-99. The simulation results substantiate the power-delay-product (PDP) benefits. For example, results for complex MCNC benchmarks indicate a PDP improvement of 61%, 56%, and 38% on average compared to three alternative techniques, evaluated at 45 nm.**

I. Introduction

Shifting from a cloud-centric approach to a thing-/data-centric perspective, known as the Internet of Things (IoT), could significantly mitigate challenges such as high latency, limited scalability, quality of service, privacy, and security. Given its promising potential, the IoT market is expected to reach $4.5 trillion by 2035, with an interconnected network of over one trillion devices encompassing smart homes, cities, industries, healthcare wearables/implants, and agriculture [1]–[3]. According to Ericsson, intelligent IoT systems could reduce carbon emissions by 3%, or 63.5 gigatons, by 2030 [4]. However, IoT devices are primarily powered by batteries, which presents critical challenges related to limited lifespan, cost, maintenance, and environmental sustainability. EnABLES forecasts that without change, global battery disposal could reach 78 million units daily by 2025 [5].

In contrast, batteryless devices can overcome these significant and inherent limitations using energy-harvested systems, which harness ambient energy sources, eliminating the need for traditional batteries. However, intermittent behaviors caused by the sources can disrupt program execution and lead to data loss and unpredictable outcomes. Therefore, advanced techniques in the normally-off (intermittent) computing domain have been formulated. These methodologies offer advantageous features such as near-zero power consumption during idle states, immediate wake-up capabilities, and robustness against power failures [6], [7]. Consequently, non-volatile (NV) components, including non-volatile memories (NVMs) and non-volatile flip-flops (NV-FFs), have received significant industry and academic attention [8]. These NV elements prevent the need for a boot-up sequence post-sleep owing to their inherent nonvolatility. Typically, in NV processors, data from all registers are offloaded to NVMs before entering a deep sleep state. Throughout this sleep phase, there is no need for a continuous power source. Upon re-energizing the processor, data are retrieved from the NVMs, and system operations recommence. Recent literature has highlighted various hardware-assisted strategies tailored explicitly for intermittent computing paradigms. The authors of [9] have replaced all traditional flip-flops with NV-FFs, whereas in [10], multiple diminutive NVM arrays are used for efficient backup and data restoration. While NVMs present the benefit of data persistence, this comes at the expense of elevated write power consumption. As a result, there is a pressing need for a holistic and systematic synthesis approach. Traditional checkpoint-based strategies are susceptible to internal and external inconsistencies in the event of a power failure. Internal inconsistencies manifest when the execution context is only partially retained in the NVM, while external inconsistencies occur when power failures happen between successive checkpoints. Previous implementations have suffered notably from the computational and energy burdens of additional middleware and checkpointing operations [11], [12]. Furthermore, these traditional methods are prone to leakage between checkpoint operations, particularly when utilizing volatile registers and flip-flops in CMOS-only datapath designs. These challenges necessitate a more comprehensive focus on life cycle energy optimization, considering intermittent power supply and data streams. Despite exhaustive studies employing conventional checkpoint methods and intricate programming paradigms to address intermittent computing pitfalls, our previous work reveals that such techniques frequently encounter performance bottlenecks and limitations in scalability [7].

This paper developed a cross-layer approach to address the stated issues, which requires effective task scheduling and power management systems. At the circuit level, power management is designed, establishing various thresholds to delineate different operational zones. At the architecture level, a systematic Design Exploration of Intermittent-Aware Computing (**`DIAC`**) methodology is devised, which minimizes overhead while maximizing forward computational capabilities. The prototyped DIAC design tool, when integrated with the proposed finite-state machine, offers the core operations of an IoT node, sense, compute, transmit, and sleep states, ensuring resilience against power interruptions. By collaborating software and hardware stages, the number of checkpoints is reduced, re-

979-8-3503-4859-0/DATE24/© 2024 EDAA

ducing both power and delay overhead. In the final phase, the system is deployed in a power-scarce environment, and its performance is evaluated using a system-level in-house framework, including a set of benchmarks.

II. Intermittent Computing Review

Ambient energy sources like solar, kinetic, etc., play a vital role in pursuing sustainable and renewable energy solutions. They offer benefits such as abundance, decreased carbon emissions, and cost savings [13]. When considering embedded systems, radio-frequency identification (RFID) is often preferred as an ambient power source for several reasons, including its self-powering mechanism, scalability, and cost-efficiency. RFID technology uniquely draws power wirelessly from its reader, a feature facilitated by electromagnetic induction [14]. Unlike conventional devices that require internal batteries or wired connections, when an RFID reader emits a signal, it induces a current in the RFID tag's antenna. This powers the tag, allowing data transmission without needing batteries or regular maintenance. Although this offers efficiency in applications such as inventory management and access control, intermittent energy bursts can cause operational interruptions, leading to data loss or inconsistent results [15]. Our research focused on designing a specialized architecture using RFID sources, making integration into embedded systems more straightforward. Based on the observed behavior of this energy source, we take into account its intermittency and variability. Our objective is to optimize energy utilization and minimize dependency on nonrenewable resources by carefully designing our system to adapt to voltage-level fluctuations.

Energy harvesting devices employ intermittent computing, where short bursts of program execution are often interrupted by power failures. Despite its challenges, many studies have explored techniques to leverage this computing style at various levels. Certain challenges must be considered during the design time; the less energy used for computation, the more tasks can be performed. Task execution times can vary, with compute-intensive tasks sometimes interrupted by power losses, potentially rendering data or computations obsolete before completion. System designers need robust language and software support to interact with sensors despite fluctuating power [16]. They also need a deep understanding of the hardware and behavior of energy harvesters. From a software-level point of view, intermittent computing involves using checkpoints and task-based programming models [17], [18]. Checkpointing enables the preservation of volatile states in non-volatile memory (NVM), allowing computation to resume after a power failure. Meanwhile, task-based systems break programs into short tasks and resume execution of the last completed task. Task-based systems often employ various memory management techniques, such as privatization and variable copying. However, checkpoint approaches may suffer from internal and external inconsistencies after each power loss, and frequent checkpointing results in performance degradation. Furthermore, there are hardware modifications and approaches based on speculation and watchdog timers to optimize data storage and restore tasks during power failures. Programming languages like Chain [19] and Mayfly [20] simplify intermittent computing by segmenting programs into tasks and managing time and memory consistency. Compiler-based approaches analyze program sections and determine the minimum number of checkpoints needed to prevent incompatibilities. The main disadvantages of this model are its incompatibility with existing libraries and code bases, its lack of importability across hardware platforms, and the burden it places on the programmer. Architecture-level techniques for intermittent computing offer various solutions to overcome the challenges of power failures and optimize energy efficiency [21]. Approximate computing techniques are used to reduce the accuracy of computations to improve performance and efficiency. Hardware solutions such as Clank track memory access to detect write-after-read sequences, while Cascaded Hierarchical Remanence Timekeeper (CHRT) provides resilient timekeeping during power failures [22].

III. Proposed Intermittent-Aware System

A. Hierarchical Design Exploration

Since the charge and discharge cycle – an intrinsic characteristic of energy harvesting devices – may occur more than hundreds of times per second, NV elements can maintain their computation state and ensure forward progress in the presence of unpredictable energy failures. Critical difficulties encountered in prior research include the requirement for intricate software that lacks scalability, performance overhead introduced by NVM components, and system inconsistencies. In this section, a co-design methodology is introduced to push the boundaries of intermittent computing. This methodology entails the development of a novel architecture and software strategy that minimizes overhead while maximizing forward progress. Figure 1 shows the design flow of our systematic Design Exploration of Intermittent-Aware Computing (**DIAC**) methodology to address power failure with minimum performance overhead, offered by three complementary procedures:

1) Tree generator: The Tree Generator takes the high-level program, in Step ❶, and synthesizes it to RTL-level Hardware Description Language (HDL), SPICE netlists, etc., and generates an un-optimized tree, where nodes contain functions and their power consumption, and edges indicate their connections. In Step ❷, we calculate power consumption using the commercial synthesis tool, including Synopsys DC and HSPICE. Then, in Step ❸, the DIAC procedure will produce a *feature dictionary* (Dict.) and a *tree-based illustration*. Each node, e.g., node *i* in level *j* (n_j^i), has one feature dictionary, which contains the number of inputs from a lower level (fan_{in}), the number of outputs to an upper level (fan_{out}), the node level itself (*j*), and its power consumption. A modified tree-based depiction of intermittency and peak harvested power (V_{peak}) will be derived using the following three strategies. *Policy1*: Large components (functions) will be broken into smaller tasks with lower power to meet the desired conditions and criteria, including $avg(F_{power}) < V_{th} << V_{peak}$. Due to the division's nature, this provides the best resiliency at the cost of performance overhead. *Policy2*: Small components will be merged into larger components with a higher power to meet the

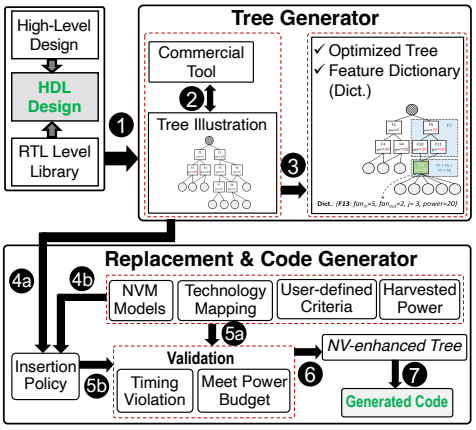

Fig. 1. Design flow of the DIAC framework.

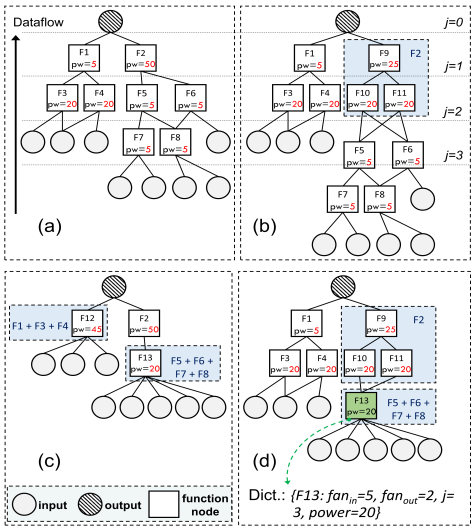

Fig. 2. Tree illustrations of an 8-input/1-output design using different approaches (a) original, (b) Policy1, (c) Policy2, and (d) Policy3.

conditions, $max(F_{power}) << V_{th}$ and $min(F_{power}) = n\%Max$, which provides best performance at the cost of lower resiliency, resulting from larger components. *Policy3*: This option is positioned between the first two policies and offers better performance and resiliency than Policies 1 and 2, respectively. Figure 2 depicts the original graph and its three representations regarding different conditions and needs.

2) Replacement: The replacement procedure aims to assess design feasibility and identify efficient replacement algorithms to address power failure, focusing on minimizing area and energy overheads. Applying *Partitioning* and *NVM Insertion* modules to the produced tree ensures attestation function integrity and forward progress under various failures. The partitioning is analogous to task-based methods existing in software approaches; however, our approach allows us to partition the netlist and ensure integrity statically. Given the modified tree from Step 4a, power budget, and NVM features from Step 4b, prioritizing nodes and finding replacement points efficiently requires weighing efficiency and resiliency. Generally, NVMs have a higher write cost than volatile memory, like SRAM, so reducing the number of NVMs' writes reduces power and delay. Three criteria define the replacement policy, realizing a more power-efficient system: (I) Nodes in the upper level: If NVMs are inserted in closer nodes to the output, the structure will be more power efficient under power failure conditions. (II) Nodes with higher power consumption: If NVMs are inserted in a node or a cone of nodes with a total higher power consumption, the system will be more power efficient. (III) Nodes with higher fan_{in} and/or fan_{out}: If a node with more inputs and/or outputs is integrated with NVMs, the total number of writes will be reduced by a factor of $1/(fan_{in} + fan_{out})$. Considering the above criteria, traversing the tree starts from leaves (bottom, inputs) upwards (roots, outputs) as follows: The total consumed power (P_{total}) until arriving at a node (n) equals the summation of all the previous nodes' power consumption. By inserting an NVM in the position, n, the previous power values are set to zero, and the node's Dict. is updated with the new power consumption = $P_{total} + P_n$. Updates will be performed in parallel for all nodes at the same level. These criteria will be assessed and modified during the execution in Step 5.

3) Code generator: After traversing all the levels and inserting NVMs, the NV-enhanced tree will be formed in 6. Finally, in Step 7, the generated code undergoes validation checks for possible timing violations. To do so, this optimized tree will first be converted to an HDL code and submitted to the commercial tool. Upon passing, the design efficacy is determined based on the proposed performance metrics models and intermittency behavior. Incorporating tree-based representations, different designs, and power failure scenarios will exponentially expand the design space. This will necessitate an efficient, precise, automated design tool that seamlessly converts any combinational and sequential designs into intermittent robust architectures without human intervention.

B. Architectural States and Transitions

Batteryless systems can be broadly categorized into two primary types. The first category can complete all computations when the battery is fully charged. Conversely, the second type has inadequate capacitor energy storage to satisfy all computations. For the second type, storing intermediate registers in NVMs is imperative. Herein, all operations, namely sense (*Se*), compute (*Cp*), transmit (*Tr*), sleep (*Sp*), and backup (*Bk*), are divided into atomic operations, which are executed uninterrupted. These atomic operations are determined based on the system's maximum storage power and should only begin when sufficient power is available. We will iteratively use three policies to determine optimal atomic operations to maximize efficiency. The finite state machine (FSM) of the system offering the core operations of an IoT node, sense, compute, transmit, and sleep states is depicted in Fig. 3 (a). Initially, the system is in *Sp* and reverts to this state after completing each atomic operation to conserve power for the next process. Integrating DIAC's produced design with the proposed FSM forms an intermittent-aware sensor node, ensuring resilience against power interruptions, shown in Fig. 3(b).

A detailed description of the required steps is illustrated in Algorithm 1. The algorithm starts with the main while looping in line 3. This loop runs according to the *interval* value in line 33. As this loop is executed, the system's state changes depending on the Reg_Flag and energy level (lines 6-11). The system has four threshold voltages for each state (Th_{State}),

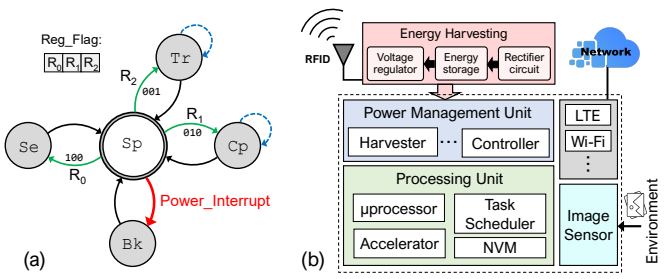

Fig. 3. (a) The proposed state machine description, and (b) an intermittent-aware sensor node.

e.g.,Th_{Cp}, along with two more thresholds Th_{Safe_Zone} and Th_{Off}. The system can perform tasks in each state if and only if criteria, including Reg_Flag and enough power to complete the desired operation ($Energy > Th_{State}$), are satisfied. Once the system transitions to either the **Cp** or **Tr**, it will stay in this state until the power drops below the threshold defined by Th_{Safe_Zone} (lines 17 and 27). The *Safe_Zone* threshold creates a narrow range that lies between the exit points of **Cp** or **Tr** and the beginning of **Bk** state. The system can harvest further energy within this range, allowing it to restore to its prior state instead of reverting to **Bk**. The transition to the next state is determined by modifying the value of the Reg_Flag. However, this progression can be halted at any state by setting the Reg_Flag to `'0b000'`. For example, in **Cp**, the Reg_Flag stays unchanged until the computation is done. Then, according to the computation result (lines 20-23), the Reg_Flag is set to **Tr** or **Sp** and exits the loop. During either **Cp** or **Tr**, when the power level falls below the *Safe_Zone*, the system reverts to **Sp** (lines 25-32). There are two interrupt routines in our system. The first interrupt is for the `Timer` (line 34), which is the maximum sampling rate of the system. However, it is important to note that this frequency can be reduced depending on the system's power. The second interrupt (`Power`) is generated by the power management unit (line 38). Suppose the total energy of the system is less than the threshold energy required for backup. In that case, the power management unit triggers an interrupt, and the backup unit stores all the necessary intermediate registers based on the register flag (Reg_Flag). Since the energy of the system reduces in the **Sp** state (due to the standby energy), the system requires an interrupt to change its state to **Bk**.

Algorithm 1 State machine of the proposed system.

```
1:  States = [Sp, Se, Cp, Tr, Bk] ▷ Sleep, Sense, Compute, Transmission, and Backup.
2:  Initialize State ← Sp
3:  while (True)
4:    if (State == Sp)
5:      Reg ← Read (Reg_Flag)
6:      if (Reg == 0b100 & energy > Th_Se)      ▷ The green arrow from Sp to Se.
7:        State ← Se
8:      elsif (Reg == 0b010 & energy > Th_Cp) ▷ The green arrow from Sp to Cp.
9:        State ← Cp
10:     elsif (Reg == 0b001 & energy > Th_Tr) ▷ The green arrow from Sp to Tr.
11:       State ← Tr
12:   if (State == Se)
13:     Sample ← Sense()
14:     State ← Sp
15:     Reg ← 0b010
16:   if (State == Cp)
17:     while( energy > Safe_Zone)              ▷ Dashed blue arrow in Cp.
18:       Result ← Compute(Sample)
19:       if (sample is completely processed)
20:         if (transmission is require)
21:           Reg ← 0b001
22:         else
23:           Reg ← 0b000
24:         Break                                ▷ Break the loop!
25:     State ← Sp                               ▷ The black arrow from Cp to Sp.
26:   if (State == Tr)
27:     while(energy > Safe_Zone)                ▷ Dashed blue arrow in Tr.
28:       Transmit (Result)
29:       if (transmission is done)
30:         Reg ← 0b000
31:         Break                                ▷ Break the loop!
32:     State ← Sp                               ▷ The black arrow from Tr to Sp.
33:   Sleep (interval)          ▷ Interval is determined by the average charging rate.
34: interrupt_Timer()           ▷ Check the sensing interval is valid or not.
35:   Reg ← Read_Reg_Flag (interval)
36:   if (Reg == 0b000)
37:     Reg ← 0b100
38: interrupt_power()           ▷ Power is scarce to perform any task!
39:   State ← Bk
40:   Backup()                  ▷ Back up all intermediate values w.r.t. register value.
41:   State ← Sp                                 ▷ The black arrow from Bk to Sp.
```

IV. Results

A. Validation of DIAC Approach

As previously noted, *Policy3* simultaneously provides acceptable resiliency and efficiency. Thus, we utilize it as a replacement policy and the principal methodology in this section. Figure 2(d) shows an example of this approach where the upper and lower limits for power consumption are set at 25mJ and 20mJ per operand, respectively. Consequently, operands consuming more than 25mJ of power must be divided, while those using less than 20mJ should be combined. Thus, F_{5-8} are merged to be represented by F_{13}. Conversely, F_2 is broken down into smaller operands, i.e., F_{9-11}. This step is followed by creating a dictionary based on the operands' number of inputs and outputs, delays, and power consumption. To accurately estimate the power and delay of each operation, DIAC harnesses the output from HSPICE, considering delay, static, and dynamic power for each operand, which comprises multiple gates. We then devised a mathematical model to meet diverse needs and estimate design-time evaluation parameters before run-time. We approximated the dynamic energy using $\approx 2 \times \sum_{i=0}^{n} delay_i \times dynamic\_power_i$, where n represents the total number of gates in an operand, and the delay is determined when both input and output equal $\frac{V_{DD}}{2}$. For a more accurate energy consumption estimation, this delay is doubled. It is essential to highlight that while one gate is in switching mode, other gates remain inactive without any activities. Therefore, the power consumption of these inactive gates is determined by the critical delay path (CDP) multiplied by the total number of gates ($\approx CDP \times \sum_{i=0}^{n-1} static\_power_i$), excluding the currently active gate. A system-level in-house framework is developed to verify our approach's functionality. Firstly, we implement a design by traversing its NV-enhanced tree generated by DIAC and concerning the system's parameters, including peripherals, using HSPICE in the 45nm NCSU PDK library. The memory controller and registers are designed and synthesized by Design Compiler. Afterward, we incorporated the results from circuit-level assessments and extensively modified CACTI at the architecture level. Next, we integrated the architecture with the proposed FSM and exported the performance to an in-house cross-layer framework, taking the CACTI output and application netlist as the inputs. At the application, we evaluated

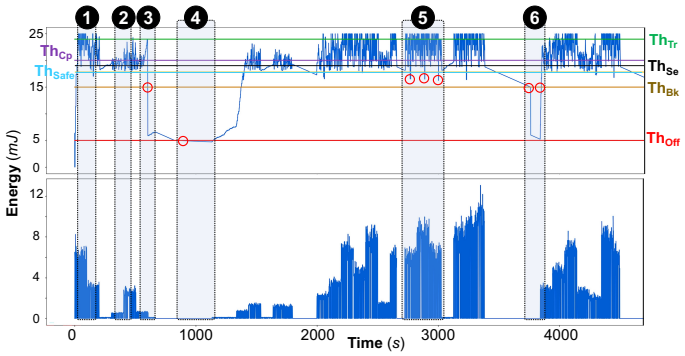

Fig. 4. Battery (E_{Batt}) (top) and charging rate (bottom) of the system.

the performance of our proposed technique in the presence of power outages using various benchmarks. To incorporate the behavior of intermittency into our study, we adopted a methodological approach that involved simulating an intermittent power source characterized by a predetermined sequence of voltage levels that cyclically repeat. To accurately represent the behavior of intermittent power, we introduced a virtual energy source within our simulation framework, designed to mimic the functionality of a battery. This virtual energy source is responsible for accumulating energy during power availability and deducting energy consumption during periods of power unavailability. Throughout the simulation process, we closely monitored several key parameters to ensure an accurate representation, including power/voltage levels, power availability, and the level of the virtual energy source. Monitoring is achieved by observing the behavior of the power source and virtual battery, as well as tracking the power consumption associated with various computational operations. In the system, a capacitance of 2mF is considered, and an operational voltage of 5V is used. Therefore, the system can store a maximum of $E_{MAX} = 25$mJ of energy. Herein, we assume that the sense, compute, and transmit operations consume 2mJ, 4mJ, and 9mJ, respectively, all with a $\pm\%10$ uncertainty. Furthermore, the Th_{Safe_Zone} region exceeds the backup threshold by 2mJ. In this zone, the system enters sleep mode without requiring a backup. However, if the battery energy drops below Th_{Bk}, the NVMs save all essential registers.

Figure 4 illustrates the energy (E_{Batt}) stored in the capacitor concerning the system's charging rate. In ❶, the charging rate surpasses the system's needs. Consequently, the energy stored occasionally reaches its maximum capacity of E_{MAX}, i.e., 25mJ. This allows the system to operate at peak performance. Conversely, in ❷, the charging rate is insufficient to meet the system's demands. The system remains in the **`Sp`** state until E_{Batt} surpasses Th_{Cp}. It then transitions to either **`Cp`** or **`Tr`** states. During this phase, the system continues operations until the energy drops below Th_{Safe_Zone}. In ❸, a sudden decline in the charging rate is noted, falling below Th_{Bk}. Consequently, the system backs up its registers to the NVMs. Following the backup in ❹, a sustained low charging rate causes E_{Batt} to drop beneath Th_{Off}, resulting in a complete system shutdown. Upon accumulating sufficient power, the system then retrieves register data based on the NVM values. As previously explained, the Th_{Safe_Zone} threshold is crucial in minimizing NVM writes. As exemplified in ❺, the system enters this zone thrice, maintaining its state in **`Sp`**. Throughout these three instances, since E_{Batt} never fell below Th_{Bk}, no energy-intensive NVM writes were needed. Subsequent efficient energy harvesting allowed the system to transition back to an active domain, fetching states directly from volatile storage and the Reg_Flag. A different scenario unfolds in ❻, where the charging source is interrupted, prompting the system to revert to the **`Sp`** state. Despite being in this state, a minimal leakage current persists, causing the system's energy to fall below Th_{Bk}. This triggers a backup process. But the charging is restored before a complete power outage, enabling the system to resume operations. Herein, there's no necessity to fetch register values from the NVMs.

B. Performance Evaluation

In this section, the developed design tool, including DIAC and the FSM, is leveraged to implement large-scale circuit benchmarks, including ISCAS-89, ITC-99, and MCNC. To evaluate DIAC performance using the developed framework, the power-delay-product (PDP) values for the four schemes are considered. The NV-based method operates similarly to conventional checkpointing, where flip-flops (FFs) are replaced by the NV-FFs to store states. It provides the highest resiliency at the cost of significant overhead. The second compared technique is NV-Clusting, presented in [7]. In this approach, the authors introduced a new concept, logic-embedded flip-flop (LE-FF), which can realize Boolean logic functions and an inherent state-holding capability. To make the evaluation more comprehensive, we have considered two DIAC-based implementations, excluding and including Th_{Safe_Zone}, denoting DIAC, and optimized_DIAC designs, respectively. According to subsection 2.B, this state allows us to reduce power consumption and delay by reducing the number of NVM writes required. Because of this, the optimized_DIAC process significantly reduces the number of NVM writes as a costly operation. It is worth noting that the safe zone varies based on the harvested energy. Figure 5 depicts the PDP results for all the mentioned approaches. For various ISCAS-89, ITC-99, and MCNC benchmark circuits, these results exhibit an average of 36% (25%), 41% (33%), and 34% (28%) PDP improvements, respectively, for the DIAC_based designs compared to NV-based (NV-clustering) implementations. Further PDP improvements are achieved by using the optimized_DIAC methodology, which provides up to 61, 56, and 38 percent average PDP improvements compared to NV-based, NV-clustering, and DIAC approaches, respectively. The benefits are achieved because of the optimal NVM write operations.

C. Conditions and Discussion

Two assumptions are considered when assessing the DIAC methodology:(1) There is never enough energy in the system to complete a process (instance). This means that using conventional CMOS-based designs without NVMs will not work. To ensure this condition, if a benchmark circuit, e.g., s27, consumes less energy than battery capacity, it is rerun multiple times till the total energy exceeds the capacity. Then, we considered all performed operations as one bigger and/or

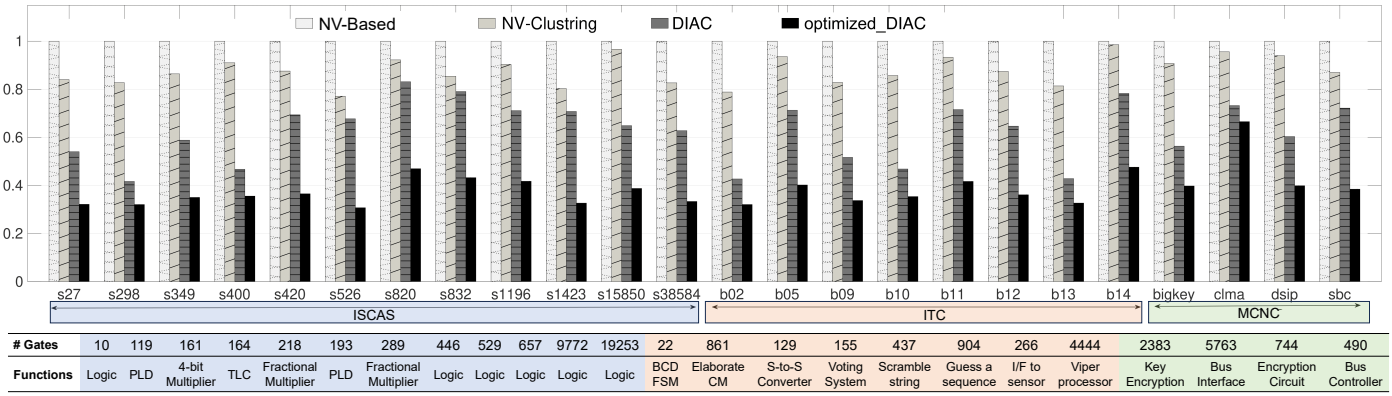

Fig. 5. Normalized PDP compared to intermittent-aware implementations for different benchmark circuits.

more complex task; (2) To make a fair comparison among different intermittent robust computing systems, the same NVM technology is leveraged. Extensive research has been conducted on designing NVMs using different NV elements, such as magnetic RAM (MRAM), resistive random access memory (ReRAM), ferroelectric random-access memory (FeRAM), and phase change memory (PCM). Due to the International Technology Roadmap for Semiconductors (ITRS) report, which identifies spintronic devices as capable post-CMOS candidates, we chose MRAM as our NVM technology. MRAM cells provide non-volatility, near-zero standby power, high integration density, and radiation-hardness features. Moreover, MTJs can be vertically integrated at the back-end CMOS fabrication process, resulting in lower interconnect energy losses and less area overhead. It is noteworthy that although varying NVM technology changes (reduces/increases) the enhancement, the overall improvement trend remains relatively stable. This is achieved because the DIAC approach optimizes NVM writes as an energy-hungry operation. For example, if ReRAMs replace MRAM cells, the optimized_DIAC exhibits higher efficiency than the other examined techniques because the ReRAM write consumes $\sim 4.4\times$ more energy than MRAM.

V. Conclusion

This paper proposed the **DIAC** methodology to enhance intermittent computing systems' efficiency. This synthesis process is intricately designed to ensure the progress of the tasks while optimizing energy consumption. Coupled with a detailed FSM that characterizes core IoT operations, the DIAC-based design proved efficiency and resiliency against power disruptions. Furthermore, a wide range of benchmark circuits has showcased the DIAC's superiority over the previous schemes, with a significant PDP improvement.

Acknowledgments

This work is supported in part by the National Science Foundation under Grant No. 2303114.

References

[1] H. Bauer *et al.*, "Internet of things: Opportunities and challenges for semiconductor companies," *Article by McKinsey'soctober*, 2015.

[2] T.-H. Hsu *et al.*, "Ai edge devices using computing-in-memory and processing-in-sensor: from system to device," in *IEEE IEDM*. IEEE, 2019, pp. 22–5.

[3] S. Liu *et al.*, "Energy-aware mac protocol for data differentiated services in sensor-cloud computing," *Journal of Cloud Computing*, vol. 9, no. 1, pp. 1–33, 2020.

[4] J. Malmodin and P. Bergmark, "Exploring the effect of ict solutions on ghg emissions in 2030," in *EnviroInfo and ICT for Sustainability 2015*. Atlantis Press, 2015, pp. 37–46.

[5] M. Hayes *et al.*, "Enables: European infrastructure powering the internet of things," in *SSI; 13th International Conference and Exhibition on Integration Issues of Miniaturized Systems*. VDE, 2019, pp. 1–8.

[6] B. Ransford and B. Lucia, "Nonvolatile Memory is a Broken Time Machine," *MSPC*, pp. 5:1–5:3, 2014.

[7] A. Roohi and R. F. DeMara, "NV-Clustering: Normally-Off Computing Using Non-Volatile Datapaths," *IEEE TC*, vol. 67, no. 7, pp. 949–959, July 2018.

[8] D. Chabi *et al.*, "Ultra low power magnetic flip-flop based on checkpointing/power gating and self-enable mechanisms," *IEEE TCASI*, vol. 61, no. 6, pp. 1755–1765, 2014.

[9] N. Sakimura *et al.*, "10.5 A 90nm 20MHz fully nonvolatile microcontroller for standby-power-critical applications," in *IEEE ISSCC*. IEEE, 2014, pp. 184–185.

[10] S. Khanna *et al.*, "An FRAM-Based Nonvolatile Logic MCU SoC Exhibiting 100% Digital State Retention at VDD = 0 V Achieving Zero Leakage With ¡ 400-ns Wakeup Time for ULP Applications," *IEEE JSSC*, vol. 49, no. 1, pp. 95–106, 2014.

[11] B. Ransford *et al.*, "Mementos: System support for long-running computation on RFID-scale devices," in *ACM Comput Archit News*, vol. 39, no. 1. ACM, 2011, pp. 159–170.

[12] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," *ACM SIGPLAN Notices*, vol. 50, no. 6, pp. 575–585, 2015.

[13] A. Akella *et al.*, "Social, economical and environmental impacts of renewable energy systems," *Renewable energy*, vol. 34, no. 2, pp. 390–396, 2009.

[14] R. M. Ferdous *et al.*, "Renewable energy harvesting for wireless sensors using passive rfid tag technology: A review," *Renewable and Sustainable Energy Reviews*, vol. 58, pp. 1114–1128, 2016.

[15] J. Eriksson *et al.*, "Mspsim–an extensible simulator for msp430-equipped sensor boards," in *EWSN, Poster/Demo session*, vol. 118, 2007.

[16] J. Hester and J. Sorber, "The future of sensing is batteryless, intermittent, and awesome," in *SenSys*, 2017, pp. 1–6.

[17] S. Umesh and S. Mittal, "A survey of techniques for intermittent computing," *Journal of Systems Architecture*, vol. 112, p. 101859, 2021.

[18] P. Singla *et al.*, "A survey and experimental analysis of checkpointing techniques for energy harvesting devices," *J. Syst. Archit.*, p. 102464, 2022.

[19] A. Colin and B. Lucia, "Chain: tasks and channels for reliable intermittent programs," in *ACM OOPSLA - SPLASH*, 2016, pp. 514–530.

[20] J. Hester *et al.*, "Timely execution on intermittently powered batteryless sensors," in *SenSys*, 2017, pp. 1–13.

[21] S. Ruffini *et al.*, "Norm: An fpga-based non-volatile memory emulation framework for intermittent computing," *ACM JETC*, vol. 18, no. 4, pp. 1–18, 2022.

[22] J. de Winkel *et al.*, "Reliable timekeeping for intermittent computing," in *ASPLOS*, 2020, pp. 53–67.