Idetic: A High-level Synthesis Approach for Enabling Long Computations on Transiently-powered ASICs

Azalia Mirhoseini, Ebrahim M. Songhori, and Farinaz Koushanfar
Dept. of Electrical and Computer Engineering, Rice University, Houston, Texas
{*azalia,ebrahim,farinaz*}@*rice.edu*

***Abstract*—We develop *Idetic*, a set of mechanisms to enable long computations on ultra-low power Application Specific Integrated Circuits (ASICs) with energy harvesting sources. We address the power transiency and unpredictability problem by optimally inserting checkpoints. Idetic targets high-level synthesis designs and automatically locates and embeds the checkpoints at the register-transfer level. We define an objective function that aims to find the checkpoints which incur minimum overhead and minimize recomputation energy cost. We develop and exploit a dynamic programming technique to solve the optimization problem. For real time operation, Idetic adaptively adjusts the checkpointing rate based on the available energy level in the system. Idetic is deployed and evaluated on cryptographic benchmark circuits. The test platform harvests RF power through an RFID-reader and stores the energy in a 3.3μF capacitor. For storage of checkpointed data, we evaluate and compare the effectiveness of various non-volatile memories including NAND Flash, PCM, and STTM. Extensive evaluations show that Idetic reliably enables execution of long computations under different source power patterns with low overhead. Our benchmark evaluations demonstrate that the area and energy overheads corresponding to the checkpoints are less than 5% and 11% respectively.**

I. Introduction

Recently, there has been an unprecedented growth in applications that demand autonomous energy supplies for reliable operation. Examples of such applications include medical implants, military, telemetry, and remote sensing tasks under harsh, hazardous, or inaccessible environments. Batteries fail to operate as a permanent source due to their limited energy capacity and lifecycle, a major drawback in scenarios where their replacement is either infeasible or very expensive. New advances in energy harvesting devices make them an attractive solution for ultra-low power applications.

Despite the fact that energy harvesting devices enjoy an apparently limitless supply of energy, there are major challenges that severely constrain their capabilities. One critical challenge is the harvested power intermittency. Short and intermittent availability of harvested energy prohibits long computations that require a contiguous block of time. The harvested energy may even be insufficient to sustain a given task. Another challenge is unpredictability of the input energy, which makes the system unreliable. Devising methods to address these challenges is vital for broadening the applications of energy harvesting devices.

Customized hardware designs are shown to be orders of magnitude more energy efficient than general-purpose processors. As a result, ASIC is a very attractive solution for energy harvesting applications. Despite the higher engineering and manufacturing cost, mass production justifies using ASIC solutions. In addition, recent advances in High-Level Synthesis (HLS) tools promises lower design cost and complexity.

In this paper, we propose *Idetic*[1], a novel automated high-level synthesis methodology that enables long computations on ultra-low power ASICs with energy harvesting sources. Idetic applies checkpointing strategies that enable gradual progress in computations by storing the current state of the process and retrieving it later when energy becomes available. Our new methods optimally locate low-overhead checkpoints that significantly reduce recomputation cost in systems with frequent power losses.

Our goal is to place the checkpoints such that the underlying application is successfully executed with minimum energy loss. To find the optimal checkpoint locations, we use the design's Control Data Flow Graph (CDFG). CDFG is an intermediate representation of the design that is generated while translating the high-level behavioral specifications of the system to the Hardware Description Language (HDL). We map the optimal checkpoint placement problem to a cost minimization objective and solve it with an efficient algorithm based on dynamic programming. We devise mechanisms that embed the checkpointing circuits within the HDL code. We also propose efficient techniques that adaptively sense the available energy and activate the checkpoints based on power consumption estimations.

The checkpointed data has to be stored on a Non-Volatile Memory (NVM) repeatedly. Thus, choosing an efficient and fast memory is important. We explore the performance of different state-of-the-art and emerging NVMs including NAND-flash memory, Phase-change memory, and Spin-transfer-torque memory.

Different checkpointing methods have been developed for software processors, which operate at the compiler level [1], [2], [3]. Software methods cannot be directly applied to

[1]Our methodologies name refers to *eidetic*, a condition in which the memory is able to recall exact details of the past events.

ASICs due to the fundamental implementation differences. High-level synthesis checkpointing methods have been developed for hardware designs to address the fault-tolerance problem [4], [5]. However, the existing methods target different objectives and constraints that make them inapplicable to our problem. For example, a design parameter in the available literature for fault-tolerant checkpointing uses a limited shared register file to store the checkpoints. Such fault-tolerant methods are not applicable to our problem since intermittent power losses erase the register contents. Our method thus addresses a fundamentally different objective, as we use NVM for data storage during power outages and recovery. Our main focus is on tolerating power failures. Additionally, we exploit state-of-the-art HLS tools that enable developing complex applications.

We deploy our techniques on cryptographic algorithms that are run on an RFID platform. As RFIDs become more and more pervasive, there is an ever-increasing demand for securing their underlying applications. Due to power source variation and intermittency, running complex cryptographic algorithms on RFID devices is very challenging. Our techniques enable the execution of the cryptographic applications with minimal constraints on supply energy availability. Our contributions are as follows:

- We propose Idetic, novel checkpointing methodologies that enable running long computations on ASICs with energy harvesting sources.
- Idetic is designed to maximally handle the power source variations in a real time system. It finds optimal checkpoint locations that incur the lowest energy and recomputation overhead.
- We develop the supporting tools for automatic insertion of the checkpointing circuits to the design's HDL file.
- We explore the memory effect on checkpointing applications by comparing the performance of different state-of-the-art and emerging NVM technologies.
- We show the applicability and effectiveness of our methods by applying them to benchmark circuits for cryptographic applications. Our experiments verify that Idetic causes very low energy, delay, and area overhead and can efficiently support various power trace patterns.

II. Related Work

Computing on batteryless energy harvesting systems has found applications in many fields such as cryptography, wireless sensor networks and habitant monitoring systems [6], [7]. A number of platforms that empower batteryless computational devices have been developed. The WISP is a platform that harvests RF energy from RFID-readers to supply its processing unit (TI-MSP430) [8]. Following WISP, several other platforms have been proposed that offer better performance in RF harvesting, storage, and peripherals; examples of which are EnHANTs [9] and UMass Moo [10].

The high energy efficiency of ultra-low power ASICs makes them a natural fit for systems with energy harvesting sources [11]. There are emerging applications of such systems in distributed sensor networks and medical devices. For example, Chen et al. recently developed an IntraOcular Pressure (IOP) sensor for eye pressure monitoring [12]. The sensor is placed inside the eye and achieves energy autonomy by harvesting solar energy through a solar cell with average power consumption of less than 10nW. It is believed that technology scaling to sub-20nm and subthreshold designs open doors for an upcoming class of ultra-low power devices powered by harvesting sources [13].

Checkpointing methods have been proposed for addressing the intermittent-energy problem at the compiler level. A number of techniques suggest saving all the available data at each checkpoint. Those methods are easy to implement but are shown to be less efficient due to the large overhead of the checkpoints [14]. Earlier work have also proposed voltage scaling techniques along with checkpointing to meet the deadlines imposed by limited power [15], [3]. Those methods are inapplicable to our problem, since we target ultra-low power devices that operate at a single voltage. For single-voltage devices, a number of program-partitioning techniques for handling power variations have been developed [16], [1]. For example, Mementos ([1]) proposes an automated method for computing on transient RF power on a TI-MSP430 microcontroller. It inserts the checkpoints at the end of the loops and function-calls during the compile time. It also uses a timer interrupt that periodically measures the source voltage and activates the checkpoint if necessary.

Software methods insert the checkpoints at the compiler level. However, hardware checkpointing requires CDFG or lower level descriptions. For example, a variable in high-level code is not necessarily a single register in HDL. Our method also takes into account parallelism/pipelining techniques in hardware while placing the checkpoints. However, software methods such as Mementos do not have access to such low level information during the compile time.

A suite of prior work has developed high-level synthesis checkpointing and rollback recovery algorithms to address the fault-tolerance problem in ASICs [4], [17]. The goal has been to insert the checkpoints to minimize either the corresponding hardware overhead for a given execution time, or to minimize the execution time for a limited hardware. The hardware constraint is imposed by a shared register file that is used to store temporary variables. The algorithms ensure that at the time of checkpointing enough empty registers are available. Such methods are not resilient to power loss as the register content would be lost. Since Idetic uses a separate NVM for checkpointing purposes, we do not have the shared register file constraint. Methods that combine checkpointing and replication techniques (redoing tasks) to achieve fault-tolerance have also been proposed [5]. However, our objective is to avoid replication; our assumption is that the source power is scarce and the device's energy consumption outpaces the harvested energy.

III. Preliminaries

In this section we briefly describe the concept of CDFG which is exploited in our checkpointing algorithm. We also discuss Idetic's target platform.

Control Data Flow Graph (CDFG): A control data flow graph is a way to visualize the flow of data through the hardware system. It models the connections and dependencies between processes. The nodes of the graph are basic-blocks that can represent operations, loops, and conditionals. The edges of the graph indicate the direction of data flow from one node to the other. We use the information provided by CDFG to form an optimization function that locates the highly efficient checkpoints. The system level design automation is being introduced as the next production boost in the semiconductor industry [18]. Several HLS tools have emerged that enable automatic synthesis of high-level specification codes such as C/C++ to Register Transfer Language (RTL) level specifications optimized for ASIC or FPGA implementations. In this work, we use AutoESL ([19]) for our high-level synthesis and implementation purposes. AutoESL automatically generates CDFG from high-level C/C++ codes.

Energy harvesting platform: Idetic targets general batteryless devices with intermittent power source. For experimental evaluations, we adopt the energy harvesting model from [10]. UMass Moo board is equipped with an antenna module that harvests RF power from an RFID reader. The harvested energy is stored in a capacitor which is the only energy source of the device.

IV. Checkpointing

Idetic applies the checkpointing method in two phases: design time and real time operation. During the design time, Idetic locates the checkpointing spots and automatically embeds the checkpointing circuit. We first find the overhead cost of checkpointing at the end of each state in terms of energy and time. Next, we apply our optimization methods to find the best checkpoints which incur minimum overhead cost. The distance between two consecutive checkpoints is limited to ensure task completion. After the checkpoints are placed, the checkpointing circuit is inserted to enable storing and retrieving data. We will show that hardware implementation of such circuits incur very low cost. During the real time operation the capacitor's voltage is sensed to decide whether or not to activate a checkpoint. This avoids unnecessary checkpoints when the power supply is sufficient. The global flow of our approach is shown in Figure 1.

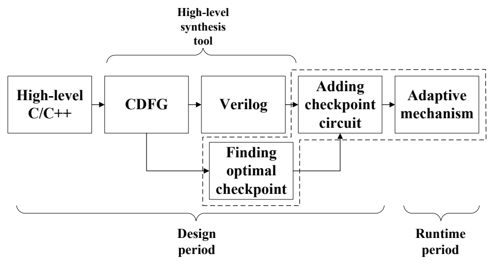
High-level synthesis tool
High-level C/C++
CDFG
Verilog
Adding checkpoint circuit
Adaptive mechanism
Finding optimal checkpoint
Design period
Runtime period

Figure 1. Global flow of our checkpointing design.

In the following, we begin with a motivational example that shows importance of checkpoint placement strategy on performance. Next, we explain how to find the checkpointing and recomputation energy overhead at different states. Finally, using the computed costs, we present our static and adaptive checkpointing strategies. We define our problem in the following format:

Objective. Enabling running lengthy applications on ASICs with energy harvesting sources.

Given. High-level synthesis design of the ASIC. The energy harvesting platform properties. In our experimental platform, we need the information about the capacitor's characteristics.

Problem. Finding optimal locations to insert the checkpoints that incur minimum energy overhead and maximally reduce recomputation energy cost in case of power failures.

A. Motivational Example

At each checkpoint location, all the information that is needed for restarting the computations from that position should be stored. Depending on the progress in the computations, the amount of data to be stored varies. Larger data makes checkpointing more costly, since more data should be written on and read from the memory. In the following example we demonstrate the importance of properly inserting the checkpoints. Figure 2 shows a CDFG with five states. The cost for processing each state has been marked on the figure. For example, the cost for completing the first state (S_1) is 2 energy units. We compare two checkpointing strategies; Strategy A inserts the checkpoints at points A_1 and A_2 and Strategy B inserts the checkpoints at points B_1 and B_2. The figure shows the checkpointing cost at each point, e.g., the cost of checkpointing at A_1 is 1 energy unit.

Now we calculate the energy loss caused by failures at different states. If the failure occurs at the end of state 1 (S_1), the loss for both strategies is 2 units since we spend 2 energy units on S_1 (which is the cost of S_1). If the failure occurs at state 2 (S_2), the energy loss for the first and second strategies are 5 and 6 units respectively. In Strategy A, since A_1 is checkpointed, we only lose the cost of S_2 which is 4 units. We also spend 1 unit for checkpointing at A_1. Thus in total we lose 4+1=5 units. In Strategy B, for a failure at S_2, we lose all resources that are spent at S_1 and S_2, that is 2+4=6 units. Using the same loss calculation method, Table

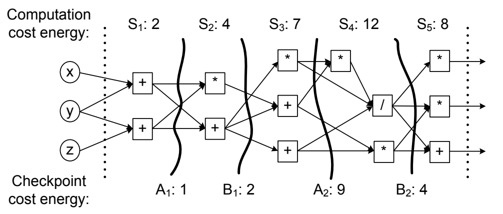

Figure 2. Different checkpointing strategies affect the performance. Computational cost of the states (S_1, S_2, ..., S_5) are shown. Checkpoints are marked as A_1 and A_2 (Strategy A), B_1 and B_2 (Strategy B) with numbers next to them indicating their cost. Strategy B outperforms Strategy A.

Table I
ENERGY LOSS IS CALCULATED FOR FAILURES AT DIFFERENT STATES.

Failure at state:	1	2	3	4	5	Sum
Strategy A's resource loss	2	5	12	22	30	71
Strategy B's resource loss	2	6	9	21	14	52

Pseudocode-1: Calculate Cost of Checkpointing.

```
1  for $e_{s_i s_j} \in E$
2      $\Phi = \emptyset$
3      for $s_v : 1 \leq v \leq j-1$
4          for $s_u : j \leq u \leq M$
5              for $b_n, b_m$ which $b_n \in s_v$ and $b_m \in s_u$
6                  if $P(b_n, b_m) = true$
7                      $\Phi = \Phi \cup b_n$
8                  if $P(b_m, b_n) = true$
9                      $\Phi = \Phi \cup b_m$
10     if $j \leq i$
11         for $s_v : j \leq v \leq i$
12             for $b_n, b_m$ which $b_n \in s_v$ and $b_m \in s_j$
13                 if $P(b_n, b_m) = true$
14                     $\Phi = \Phi \cup b_n$
15     $Cost(e_{s_i s_j}) = 0$
16     for $b \in \Phi$
17         $Cost(e_{s_i s_j}) = Cost(e_{s_i s_j}) + width(b)$
```

I shows the resource loss for failures at different states for the two strategies. Assuming that the failures happen with equal probability, Strategy B outperforms Strategy A since the sum of the losses incurred by Strategy A is 71 units while it is 52 units for Strategy B. The reason behind this is that the total cost of checkpointing at B_1 and B_2 is less than that of A_1 and A_2. In addition, checkpoints B_1 and B_2 are inserted at the end of the states that consume more energy.

B. *Computing cost function*

Idetic measures the checkpoints energy cost as well as the computational energy at different states for finding the ideal checkpoint locations. We exploit CDFG output files from HLS tools. The outputs also provide information about the Finite-State Machine (FSM) of the design. Figure 3 shows an example CDFG and its corresponding FSM.

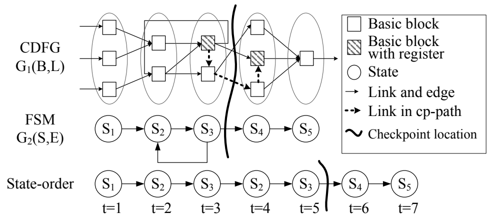

Figure 3. CDFG, FSM, and State-order of the design. An example checkpoint is inserted between s_3 and s_4.

We define a set of notations as follows. We denote a graph corresponding to the CDFG by $G_1(B, L)$, where $B = \{b_1, ..., b_N\}$ is the set of basic-blocks (e.g., adder, multiplier, condition) and L is the set of links (edges) indicating the flow of data through the basic-blocks (nodes) in the CDFG. FSM graph is denoted by $G_2(S, E)$ where $S = \{s_1, ..., s_M\}$ is the set of states (nodes) and E is set of edges showing the transitions between the states in FSM. S is a partition of the set B; each node in B belongs to exactly one of the states in S. Some basic-blocks have registers that can save outputs for future clock cycles. To keep track of registers needed for checkpointing, we define a Boolean function $P(b_n, b_m)$ for $b_n, b_m \in B$. The function's output is true when the following conditions hold: first, b_n has a register; second, there is a path between b_n and b_m such that non of the basic-blocks in the path (except b_n and b_m) has registers. If such a path exists, the value of the register b_n should be stored for recovery. We refer to the path as a *cp-path*. A sample cp-path is shown in Figure 3.

The overhead cost of checkpointing at a state is determined by the amount of registers needed to be stored for recovery which is calculated in Pseudocode-1. We denote by $Cost(e_{s_i s_j})$ the number of bits needed for checkpointing at state s_j given that the previous state was s_i. First, we list all basic-blocks that should be stored for checkpointing in a set Φ. For an edge $e_{s_i s_j}$ in E, we initially set Φ as empty (Line 1-2). Next, we add to Φ all the basic-blocks of the cp-paths which start before s_j and end after s_j or vice versa (Line 3-9). If $e_{s_i s_j}$ is a loop edge (such as $e_{s_3 s_2}$ in Figure 3), we also add all the basic-blocks in the cp-paths starting at a state between s_j and s_i and ending at s_j (Line 10-14). Finally, to calculate the cost of checkpointing, we add the width of all registers in Φ (Line 15-17). The overhead $Cost(e_{s_i s_j})$ is converted to energy cost based on the properties of the underlying NVM (Table III). For finding the computational energy cost at each state, we performed power simulations using Synopsys Design Compiler.

C. *Checkpointing Methods*

For finding the optimal checkpoint locations, Idetic needs to know the order of execution of states which we refer to as *state-order*. State-order is derived by unrolling the FSM and turning it into an acyclic sequence of states. As can be observed in Figure 3, the state-order keeps the state number

Pseudocode-2: Dynamic programming for minimizing C_{OF}

```
1  for $t : 0 \leq t \leq T$
2      for $k : 0 \leq t \leq K_{max}$
3          $C_{OF}(t,k) = +\infty$
4  for $t : 0 \leq t \leq D_{max}$
5      $C_{OF}(t,0) = -C_{CO}(t)$
6  for $k : 1 \leq k \leq K_{max}$
7      for $t : k \leq t \leq T$
8          $C_{OF}(t,k) = C_{CP}(t) + \min_{1 \leq i \leq D_{max}}$
9          $\{C_{OF}(t-i,k-1) - C_{CO}(t) + C_{CO}(t-i)\}$
```

at each clock cycle. We denote the length of the state-order by T (which is 5 in the example). We performed RTL-level simulations in ModelSim to find the state-order.

Dynamic programming: Our algorithm exploits Dynamic Programming to place the checkpoints such that they satisfy the following two conditions: (i) the checkpoints incur minimum energy overhead (ii) the maximum distance between two consecutive checkpoints is limited. Table II defines the parameters that are used in our algorithms. The objective is to insert the checkpoints such that the overall energy to finish all the T states in the state-order, is minimized. By the overall energy, we refer to the computation and the checkpoint overhead energy. The Objective Function (OF) is denoted by $C_{OF}(t,k)$ which is the overall energy for completing k checkpoints at the end of the t^{th} state of the state-order. The cost $C_{CP}(t)$ corresponds to $Cost(e_{s_i s_j})$, where s_i is the $(t-1)^{th}$ state and s_j is the t^{th} state of the state-order. The cost $C_{CO}(t)$ is the sum of the computation energy consumptions of all the states from the beginning to the t^{th} state in the state-order.

Table II
DEFINING PARAMETERS.

T	Length of the state-order
$C_{CP}(t)$	Energy consumption for checkpointing at the t^{th} state in the state-order
$C_{CO}(t)$	Energy consumption for running the application until t^{th} state in the state-order
D_{max}	Maximum number of states between two consecutive checkpoints in the state-order

We exploit dynamic programming to find the checkpoint locations. The algorithm is shown in Pseudocode-2. The initial conditions are set to ensure that the first checkpoint is located at a maximum distance of D_{max} from the beginning (Line 1-5). By K_{max}, we denote an upper bound on the number of checkpoints which satisfies the following condition: $K_{max} > \frac{T}{D_{max}}$. The minimum OF cost for inserting the k^{th} checkpoint at t, includes the cost of checkpointing at that point (Line 8) plus the minimum overall energy that is consumed before the t^{th} state (Line 9).

Periodic checkpointing: To have a comparison basis for our dynamic programming-based algorithm, we also applied a periodic approach for inserting the checkpoints. In the periodic method, the checkpoints are inserted at equal distances from each other. The periodic checkpointing does not take into account the cost of checkpointing and computational energy loss while locating the checkpoints.

Adaptive checkpointing: In hardware checkpointing applications, as opposed to software, the checkpointing circuits are located during the design time. However, a preemptive checkpointing strategy might not be efficient in scenarios that input energy is sufficient. To cope with the source energy variations, we propose an adaptive online mechanism that senses the available stored energy before each checkpoint by an Analog to Digital convertor which reads the capacitor's voltage. A checkpoint is skipped if the capacitor's energy level (corresponding to V_C) is greater than the maximum energy consumption between two consecutive checkpoints; in our target applications, this value (corresponding to V_{th}) is measured offline by power simulation. Otherwise, we complete the predetermined checkpoints and turn off the device to reduce the recomputation cost, Figure 4. We assume that the average input power is much less than the average application power consumption. This assumption is realistic for several energy scavenging sources with limited energy capacities and charging rates.

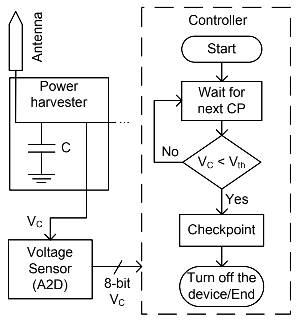

Figure 4. Adaptive sensing mechanism.

V. EVALUATIONS

In this section, first we describe the evaluation platform, including the energy harvesting model and power traces. Next, we explain in details how the checkpointing circuits are added to the register-transfer level. We also discuss our evaluation benchmarks. Finally, we provide our evaluation results that report various time, energy, and area measurements to verify Idetic's applicability and effectiveness.

A. Evaluation platform

We use Xilinx HLS tool, AutoESL, to obtain CDFG and Verilog code from high-level C/C++ source. As discussed in Section IV-C, we also need to find the state-order to have runtime information of the design. The state-order is produced by simulating the Verilog codes in ModelSim.

Table III
NVM PROPERTIES BASED ON [21], [22].

	NAND Flash	PCM	STTM
Read energy (nJ/cell)	1.5	1	0.2
Write energy (nJ/cell)	17.5	6	1.6
Read latency (ns/cell)	6.2	0.8	0.4
Write latency (ns/cell)	125	15	7
Density	1×	1-2.5×	3.75-16×

We have used Synopsis Design Compiler with FreePDK 45nm library ([20]) to evaluate power consumption and area overhead. We followed the setup shown in Figure 5 for locating and embedding the checkpoints at the RTL. AutoESL compiles only a subset of standard C/C++ for hardware implementation. Thus, we modify the benchmark source code structure accordingly.

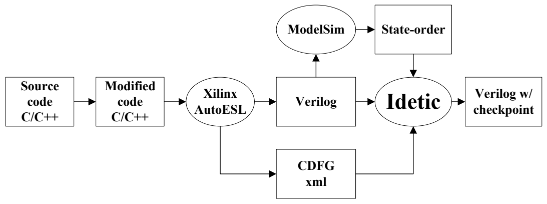

Figure 5. Experiments flow.

In our energy harvesting platform, V$_{on}$ is the capacitor's voltage at which the device turns on after a power failure and V$_{off}$ is the minimum operating voltage. We denote by $I_{leakage}$, the leakage current of the device which is determined by the power simulation. The nominal values in our model are: V$_{on}$=5.4V, V$_{off}$=3V, and the storage capacity is 3.3μF (except for RSA benchmark which is 10μF). Thus, the amount of available energy in the period from V$_{on}$ to V$_{off}$ equals 33.26μJ (100.8μJ for RSA benchmark).

To enable retrieving information after a power failure, the checkpointed data should be saved on an NVM. The energy and time for writing data on the memory affects the performance of our methods. A variety of NVMs with different characteristics have been developed. While NAND-Flash is the state-of-the-art high performance NVM, Phase-Change Memory (PCM) and Spin-Torque-Transfer Memory (STTM) are two emerging NVM technologies that exhibit better energy and speed performances. The properties of these memories are shown in Table III.

B. Power traces

We have generated a set of real power traces using UMass Moo board by varying its location with respect to the RFID reader. We have also generated synthetic power traces with different mean power and patterns. The power traces include impulses with amplitudes taken from a Gaussian distribution and inter-arrival time taken from a Poisson distribution. We will show that when the average source power is much less than the power consumption, the source pattern does not considerably affect the checkpointing results.

C. Checkpointing circuit

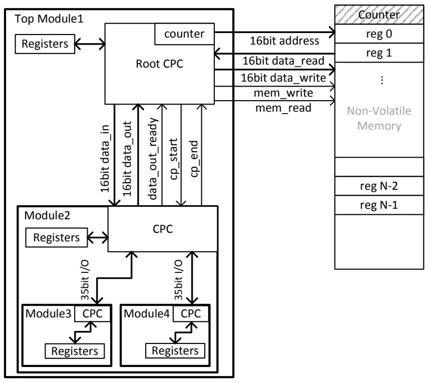

Figure 6. CheckPointing Circuit (CPC) tree structure; 35-bit bus connects the nodes of the CPC tree. Root CPC is connected to NVM.

For an input high-level source code, AutoESL generates separate modules for each C/C++ function in Verilog. There is also a module instantiation for each C/C++ function-call in Verilog description. This creates a hierarchical structure of modules. Since the CheckPointing Circuit (CPC) must have access to sub-modules' registers, the number of modules' I/Os tremendously increases. To avoid the design complexity, we implement a distributed CPC architecture. In this architecture, the CPC in each module is recursively connected to its child CPCs in sub-modules. This structures can be regarded as a tree whose root is in the top-module. The root is the only CPC that has a memory controller circuit and connects to the NVM, Figure 6.

At the time of checkpointing, the root CPC receives a permission to start checkpointing. Before starting its own checkpointing, the root gives permission to its child CPCs according to a Depth First order. It means that the CPCs which receive the permission, recursively pass it to their child nodes before sending their own data. The data passes through CPCs towards the root CPC and the memory bus. A counter at the root CPC keeps track of the number of words to be stored at each checkpoint. The counter also serves as the address of the stored words. After storing all the required registers, the counter value will be stored on the memory, Figure 6. Checkpointing procedure is completed when the root CPC sends its own data on the memory bus. A reverse procedure is used for recovering data from the NVM.

D. Benchmarks

Idetic is evaluated on cryptographic benchmark circuits including: RSA ([23]), a public-key cryptography; AES ([24]), a symmetric-key cryptography; and cryptographic hash functions MD5 ([25]), SHA1, SHA256, and all the five SHA3 round-3 candidates (BLAKE, GrØstl, JH, Keccak, and Skein)[24]. Our RSA benchmark encrypts 1024-bit data under a 1024-bit public key and a constant exponent (65537). AES benchmark encrypts 128KB data under a 128-bit key. Hash benchmarks map 128KB input to their message digest. In cryptographic algorithms, the unrolled state-orders are independent of the inputs in order to prevent side channel attacks. Thus, we plug in random inputs to extract the state-order of our cryptographic benchmarks.

E. Evaluation results

The total energy consumption for completing an application is the sum of the computation and the overhead energy. The overhead energy is the sum of recomputation energy and the energy of read/write operations on NVM for checkpointing. We define *normalized energy*, a metric that measures the ratio of the total consumed energy to the computation energy consumption. Likewise, we define *normalized time* to measure the time overhead.

We study the relationship between number of checkpoints and the normalized energy and time in Figure 8. The experiment is done on MD5 benchmark, using dynamic-programming method. The NVM type is PCM. Varying D_{max} in the dynamic method results in different number of checkpoints. The simulations are run for ten different synthetic power traces whose average power are 0.2% of the MD5 computation power. The average results are reported. As the number of checkpoints increases, due to overhead of checkpointing, the normalized energy and time values increase. For very small number of checkpoints, the normalized energy and time values grow dramatically. The reason is that a lower number of checkpoints increases the recomputation overhead due to insufficient checkpoints. The optimal number of checkpoints for MD5 is 4 for our platform. Figure 7 shows the capacitor voltage over time for completing MD5. The 4 checkpoint locations and the corresponding computational progresses can be observed. If checkpoints are not inserted, only about 22.8% of the computation will be completed before the power failure (when the voltage drops down to V_{off}) and this scenario will be repeated again next time the capacitor is charged.

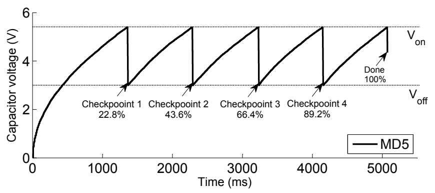

Figure 7. Checkpoint locations and corresponding progresses for MD5.

We explore the effect of different power source patterns on Idetic's performance. We run AES, MD5, and SHA256 benchmarks with four synthetic (S1-S4) and four real power traces (S5-S8). Figure 9 reports the corresponding normalized energy values. The average power of all the traces are

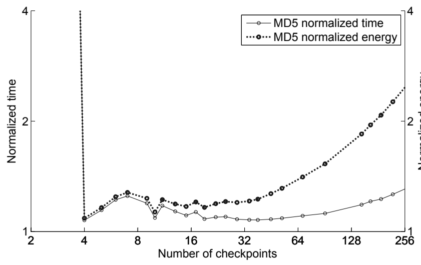

Figure 8. MD5 performance for different number of checkpoints.

3.5μW which is at least 2 orders of magnitude less than average power consumption of AES, MD5, and SHA256. As can be seen on the figure, normalized energy values of all three benchmarks do not considerably change over the synthetic and real power traces. The slight variation in the performance results arises from different amounts of energy received during the capacitor's discharge period. In scenarios, where the rate of energy consumption dominates the rate of energy harvesting, such as our target applications, the traces pattern do not significantly affect the capacitor discharge period. Thus, one can apply our algorithms to find the location of the checkpoints solely based on device specifications and the application.

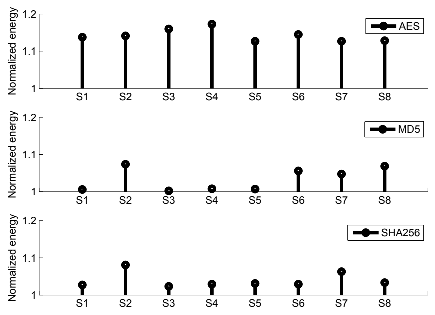

Figure 9. AES, MD5, and SHA256 normalized energy for different power traces with equal average power and various patterns.

To verify the effectiveness of our dynamic programming algorithm, we compare it with the periodic checkpointing approach. Figure 10 illustrates the normalized energy values for the two methods versus the number of checkpoints. The target benchmark is SHA256. The average result for ten synthetic source power traces are reported. For each trace the mean power values is 0.2% of SHA256's power consumption. When the number of checkpoints is around the optimal number (3), the dynamic method significantly outperforms the periodic method and introduces much lower overhead. For larger number of checkpoints, the overhead of both methods converge due to the high density of checkpoints.

We study our adaptive mechanism's performance under different source power conditions. We generate synthetic power traces with different mean power values that range between 0.2% and 300% of SHA256 average power con-

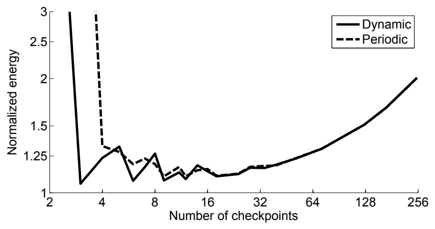

Figure 10. Performance of dynamic and periodic methods on SHA256.

sumption (which is 1.85mW in our experiments). In Figure 11, we show the normalized energy results for applying the dynamic-only and dynamic-adaptive methods on SHA256. Both methods are simulated for the optimal number of checkpoint (3). The adaptive mechanism outperforms the non-adaptive method for all tested power traces.

Idetic finds optimal number of checkpoints for scenarios where the average harvested power is much less than the power consumption. Thus, the discharge duration of the capacitor only depends on the application's energy consumption. However, as the energy harvesting rate accelerates, the capacitor's discharge duration increases. As a result, even after completing a checkpoint, the capacitor has enough energy to continue running the application. If the capacitor's energy does not last until the next checkpoints, the cost of recomputation increases. This effect explains the deep gap between the two methods performances in Figure 11. In Figure 12, we show the voltage behavior of the capacitor over time, when the average source power is around 25.4% of the application power consumption. The figure shows that the adaptive mechanism completes the application almost 1.3X faster than the non-adaptive one. For a larger input power the capacitor's energy will last until the next checkpoint, resulting in reducing the gap between the two methods.

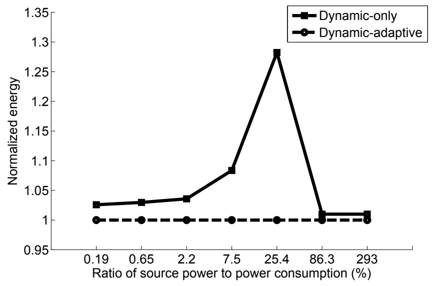

Figure 11. Comparison of dynamic-only and dynamic-adaptive mechanisms on SHA256 for different mean source powers.

To evaluate NVM effect, we measure the corresponding normalized energy and time metrics versus the number of checkpoints in Figure 13. The target benchmark is SHA256 and dynamic-adaptive algorithm is applied. The average result for ten synthetic source power traces are reported. For each trace the mean power values is 0.2% of the power consumption of SHA256. When the number of checkpoints

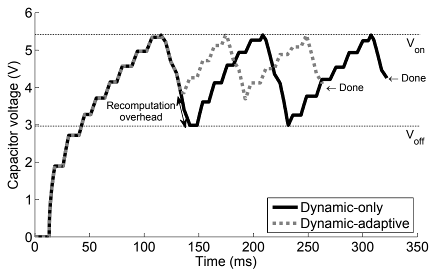

Figure 12. Simulated capacitor voltage for dynamic-only and dynamic-adaptive methods as SHA256 is running.

is low, the memory performance has an insignificant effect on the overall efficiency. This is because the overhead of the checkpoints is small. However, for larger number of checkpoints the memory characteristics affects the overhead considerably. Thus, memory selection matters only when the overhead introduced by the checkpoints is notable.

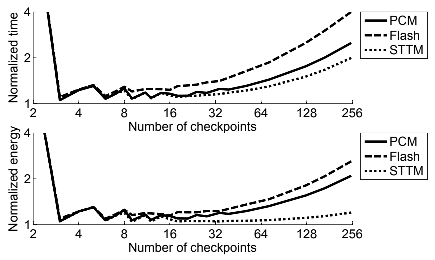

Figure 13. SHA256 performance for PCM, Flash, and STTM.

Table IV shows optimal number of checkpoints and corresponding time, energy, and area overheads for different benchmarks. The memory type is PCM. The capacitor is 3.3μF (10μF for RSA). The checkpoints incur an energy overhead of less than 11%, time overhead of less than 16%, and area overhead of less than 5%. The results verify effectiveness and applicability of Idetic. For example, the optimal number of checkpoints is 83 for RSA benchmark, which means the capacitor should be charged/discharged approximately 84 times to complete RSA. The capacity required for running RSA continuously is 84 times more than our model's capacity (10μF). Thus, our methods enable running a much longer computation on a small capacitor.

VI. Conclusion

In this paper we addressed the problem of power intermittency in ASICs with energy harvesting sources. We devised Idetic, a checkpoint placement tool that enables executing long applications by making gradual progresses as the power becomes available. Our tool adaptively adjusts the checkpoints to the power source variations and locates the ones with minimum overhead. Idetic automatically takes the high-level synthesis design as the input and outputs the

Table IV
OVERHEADS OF THE DYNAMIC METHOD ON THE BENCHMARKS

Benchmarks	optimal # of check-points	Time over-head(%)	Energy over-head(%)	Area over-head(%)
RSA*	83	1.1	1.1	1.2
AES	3	0.4	1.4	4.5
MD5	4	5.0	5.7	3.2
SHA1	1	0.4	0.2	2.5
SHA256	3	1.8	1.8	2.1
BLAKE	1	0.2	0.6	1.5
Grøstl	3	8.2	9.1	3.0
JH	13	11.0	10.3	1.2
Keccak	7	15.7	10.9	1.2
Skein	1	0.5	0.6	5.0

Verilog description of the design with embedded checkpoints. Our experiments targeted cryptographic benchmarks. We evaluated Idetic's performance for different power source conditions. The results verified Idetic's effectiveness in recomputation cost reduction caused by the power outages. The overhead associated with the checkpoints was shown to be very low: less than 16% energy overhead, less than 11% time overhead, and less than 5% area overhead were reported for our benchmarks.

REFERENCES

[1] B. Ransford, J. Sorber, and K. Fu, "Mementos: system support for long-running computation on RFID-scale devices," in *ASPLOS*, ser. ASPLOS '11, 2011, pp. 159 –170.

[2] R. Barbosa and J. Karlsson, "On the integrity of lightweight checkpoints," in *HASE*, 2008, pp. 125 –134.

[3] Y. Zhang and K. Chakrabarty, "Energy-aware adaptive checkpointing in embedded real-time systems," in *DATE*, 2003, pp. 918 –923.

[4] D. Blough, F. Kurdahi, and S. Ohm, "Optimal recovery point insertion for high-level synthesis of recoverable microarchitectures," in *FTC*, 1992, pp. 50 –59.

[5] V. Izosimov, P. Pop, P. Eles, and Z. Peng, "Synthesis of faulttolerant embedded systems with checkpointing and replication," in *DELTA*, 2006, pp. 440 –447.

[6] A. Kansal and M. Srivastava, "An environmental energy harvesting framework for sensor networks," in *ISLPED*, 2003, pp. 481 –486.

[7] C. Vigorito, D. Ganesan, and A. Barto, "Adaptive control of duty cycling in energy-harvesting wireless sensor networks," in *SECON*, 2007, pp. 21 –30.

[8] A. Sample, D. Yeager, P. Powledge, A. Mamishev, and J. Smith, "Design of an RFID-based battery-free programmable sensing platform," *TIM*, pp. 2608 –2615, 2008.

[9] M. Gorlatova, P. Kinget, I. Kymissis, D. Rubenstein, X. Wang, and G. Zussman, "Challenge: ultra-low-power energy-harvesting active networked tags (EnHANTs)," in *MobiCom*, 2009, pp. 253–260.

[10] H. Zhang, J. Gummeson, B. Ransford, and K. Fu, "Moo: A batteryless computational RFID and sensing platform." Department of Computer Science, University of Massachusetts Amherst., Tech. Rep., 2011.

[11] A. Chandrakasan, D. Daly, J. Kwong, and Y. Ramadass, "Next generation micro-power systems," in *VLSI Circuits*, 2008, pp. 2 –5.

[12] G. Chen, H. Ghaed, R. Haque, M. Wieckowski, Y. Kim, G. Kim, D. Fick, D. Kim, M. Seok, K. Wise, D. Blaauw, and D. Sylvester, "A cubic-millimeter energy-autonomous wireless intraocular pressure monitor," in *ISSCC*, 2011, pp. 310 –312.

[13] B. Calhoun, A. Wang, and A. Chandrakasan, "Modeling and sizing for minimum energy operation in subthreshold circuits," *SSC*, pp. 1778 –1786, 2005.

[14] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *LCN*, 2004, pp. 455 –462.

[15] B. Ransford, S. Clark, M. Salajegheh, and K. Fu, "Getting things done on computational RFIDs with energy-aware checkpointing and voltage-aware scheduling," in *HotPower*, 2008, pp. 5 –5.

[16] M. Buettner, B. Greenstein, D. Wetherall, and J. Smith, "Revisiting smart dust with RFID sensor networks," 2008.

[17] A. Orailoglu and R. Karri, "Coactive scheduling and checkpoint determination during high level synthesis of self-recovering microarchitectures," *TVLSI*, pp. 304 –311, 1994.

[18] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *TCAD*, pp. 473 –491, 2011.

[19] "Autoesl," 2012. [Online]. Available: http://www.xilinx.com/products/design-tools/autoesl/

[20] J. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. Davis, P. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "Freepdk: An open-source variation-aware design kit," in *MSE*, 2007, pp. 173 –174.

[21] A. Caulfield, J. Coburn, T. Mollov, A. De, A. Akel, J. He, A. Jagatheesan, R. Gupta, A. Snavely, and S. Swanson, "Understanding the impact of emerging non-volatile memories on high-performance, IO-intensive computing," in *SC*, 2010, pp. 1 –11.

[22] S. Chen, P. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," in *CIDR*, 2011, pp. 1 –11.

[23] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *CACM*, pp. 120 –126, 1978.

[24] N. I. of Standards and Technology. (2012) Cryptographic technology. [Online]. Available: http://csrc.nist.gov/groups/ST/index.html

[25] R. Rivest, "The md5 message-digest algorithm," *RFC 1321*, 1992.