# **Chime**: **Ch**eckpointing long computations on inter**m**ittently **e**nergized IoT devices

Azalia Mirhoseini, *Member, IEEE,* Bita Darvish Rouhani, *Student Member, IEEE,*
Ebrahim Songhori, *Student Member, IEEE,* and Farinaz Koushanfar, *Member, IEEE*

**Abstract**—We develop *Chime*, a set of mechanisms and methodologies that enable long-running computations on low power IoT devices with scarce and intermittent energy sources. We address the power transiency and unpredictability problem by optimally inserting checkpoints that save the intermediate states. Chime automatically locates and embeds checkpoints at the register-transfer level. We define an objective function that aims to find low-overhead checkpoints which minimize the recomputation energy cost. We develop and exploit a dynamic programming technique to solve the optimization problem. For real time operation, Chime adaptively adjusts the checkpointing rate based on the available energy level in the system. Chime is deployed and evaluated on algebraic, data transformation, and cryptographic benchmark circuits. For storage of checkpoint data, we evaluate and compare the effectiveness of various non-volatile memories including NAND Flash, PCM, and STTM. Extensive evaluations show that Chime 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 9% and 11% respectively.

## 1 Introduction

Recent years have witnessed an unprecedented growth in applications that demand autonomous energy supplies for reliable operation. Prominent and highly in-demand examples of devices that benefit from autonomous energy supplies are Internet of Things (IoT) entities including medical implants and sensors used in military, telemetry, smart building, and remote sensing applications. The bounded capacity and rechargeability of conventional batteries make them unsuitable for scenarios where their replacement is very costly or even infeasible. Energy harvesting devices are promising alternatives for batteries, as they enable self-sustained and durable energy sources.

While energy harvesting sources enjoy a seemingly unlimited supply of energy, their applicability becomes limited due to at least three sets of challenges. The first challenge arises from the non-continuous behavior of the harvested power. Short and intermittent availability of harvested energy prohibits long computations that require power availability for a contiguous block of time. The second challenge is the unpredictability of the input energy, which makes the system unreliable. The third challenge is related to the limited size of the energy storage units in many (ultra) low power applications. In this case, even when the energy storage device is fully charged, it might be insufficient to sustain a given task. Devising methods to address these challenges is vital for broadening the applications of energy harvesting devices.

In this paper, we propose *Chime*, a set of automated energy optimization methodologies and supported system designs for enabling long computations on low power IoT entities with intermittent energy sources. Our key idea is to insert efficient energy-aware checkpoints that enable gradual progress in computations by storing the current state of the process and retrieving it later when more energy becomes available. Chime optimally locates low-overhead checkpoints that significantly reduce recomputation cost in systems with frequent power losses. Our approach enables running complex computations on IoT devices in scenarios where the on-chip energy storage capacity, even if completely full, is insufficient to supply the entire workload.

Our solutions target Application-Specific Integrated Circuits (ASICs). Customized hardware designs are known to be orders of magnitudes more energy and area efficient than general-purpose processors. This property makes ASIC an attractive solution for low-power and small-scale energy harvesting applications such as IoT. Despite the higher engineering and manufacturing cost, mass production justifies designing ASIC solutions. Moreover, advances in High Level Synthesis (HLS) tools promises significant reductions in design cost and complexity [1].

To find the proper locations for inserting the checkpoints, 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). Using the CDFG, we provide two optimizations methodologies to optimally find the checkpoints that incur minimum storage and recomputing energy overheads. The first one is tailored for input-independent CDFGs, whereas the second one is suited for general input-dependent CDFGs. 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 (thereafter denoted by CPCs) within the high-level functional description. We also propose efficient techniques that adaptively sense the available energy and activate the checkpoints based on power consumption estimations.

Many fabricated Integrated Circuits (ICs) are equipped

- *Azalia Mirhoseini and Ebrahim Songhori are with the Department of Electrical and Computer Engineering (ECE) at Rice University. Farinaz Koushanfar and Bita Darvish Rouhani are with the Department of ECE at The University of California, San Diego.*

with a Joint Test Action Group (JTAG) port for post fabrication testing and debugging [2]. JTAG is used as the primary means of accessing subparts of many such ICs. An important property of JTAG is that it enables data transfer into internal Non-Volatile Memory (NVM) of the device. We present a novel approach that can take advantage of pre-existing JTAG circuitry in an IC for checkpointing, eliminating the need to modify the IoT device during design time.

The checkpointed data has to be stored on non-volatile memory 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 (PCM), and Spin-Transfer-Torque Memory (STTM).

We demonstrate the low energy overhead of Chime's checkpointing techniques. In some scenarios, however, the power requirement for writing data to NVM can be several times higher than that of the computation itself. Thus, the lower power energy source designed for the IoT device might be unable to efficiently perform checkpointing. To mitigate this challenge, we propose an efficient and customized energy supply management module for the checkpointing circuit. Our design benefits from heterogeneous capacitors that enable buffering and transferring the required electric charge from the source to the checkpointing circuit while reducing the leakage power.

Different checkpointing methods have been developed for software processors, which operate at the compiler level [3], [4], [5], [6]. Software methods cannot be directly applied to ASICs due to the fundamental implementation differences. High-level synthesis checkpointing methods have been developed for hardware designs to address the fault-tolerance problem [7], [8]. However, the existing methods target a different objective and set of 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-tolerance methods are not applicable to our problem since the 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.

To motivate and test our approach, we deploy our techniques on an RFID platform running various algebraic, data transformation, and cryptographic algorithms. 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 computationally complex algorithms on RFID devices is very challenging. Our techniques enable the execution of such algorithms with minimal constraints on supply energy availability. Our contributions are as follows:

- We propose Chime, a set of novel checkpointing methodologies that enable running long computations on IoT devices with intermittent energy sources. Our approach finds optimal checkpoint locations that incur the lowest energy and recomputation overhead.
- We design a highly efficient electric charge buffering and transfer module that supports the high power demand of the checkpointing circuit.
- We develop the supporting tools for automatic insertion of the checkpointing circuits to the design's HDL file. We also propose methods for enabling checkpointing using the existing JTAG circuitry available in many ICs.
- We explore the effect of memory in checkpointing applications by comparing the performance of different state-of-the-art and emerging NVM technologies.
- We show the applicability and effectiveness of our checkpointing methods by applying them to various benchmark circuits for algebraic, data transformation, and cryptographic applications. Our experiments verify that Chime incurs very low energy, time, and area overhead and can efficiently support various power trace patterns.

Our earlier results have appeared in [9], [10]. In this paper, we extend our contributions by (i) proposing a novel energy supply module for CPC. This module enables Chime to efficiently perform charge buffering to support the high power demand of the checkpointing circuit; (ii) devising low-overhead methods for checkpointing using the existing JTAG circuitry available in many ICs for testing and debugging purposes. Taking advantage of pre-existing JTAG for checkpointing purpose, eliminates the need to modify the IoT device during design time; (iii) performing various experimental evaluations to compare Chime's input-dependent and input-independent optimization approaches.

## 2 Related Work

Computing on batteryless energy harvesting systems has found application in many fields such as cryptography, wireless sensor networks and habitant monitoring systems [11], [12]. 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) [13]. Following WISP, several other platforms have been proposed that offer better performance in RF harvesting, storage, and peripherals; examples of which are EnHANTs [14] and UMass Moo [15].

The high energy efficiency of ultra-low power ASICs makes them a natural fit for systems with energy harvesting sources [16]. 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 [17]. 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 sub-threshold designs open doors for an upcoming class of ultra-low power devices powered by harvesting sources [18].

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 [19]. Earlier work have also proposed voltage scaling techniques along with checkpointing to meet the deadlines imposed by limited power [5], [20]. 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 [3], [21]. For example, Mementos [3] 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 [7], [22]. 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 Chime 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 [8]. 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.

Another line of work focuses on addressing the limitations of the energy harvesting powered IoTs via the concept of Non-Volatile Processors (NVP) [23], [24]. An NVP implements a set of distributed non-volatile flip-flops for backing up regular register contents. In theory, an NVP can backup the data in every flip-flop in parallel so it has the potential to reduce the sleep and wake-up time dramatically. We emphasize that Chime targets the broad and increasingly popular HLS based ASIC designs, while NVP designs are currently limited and not yet commercially widespread. However, future directions can explore a hybrid design that integrates the proposed optimization methodologies in Chime and the fast data saving and wake-up time in NVPs for more efficient checkpointing circuits. The NVPs can also benefit from the Chime checkpointing strategies in order to reduce the number of non-volatile flip-flops used for saving the state of data.

NVPs have motivated the inception of storage-less and converter-less energy harvesting systems that directly supply the harvested energy to pertinent IoT devices [25], [26]. For realizing this technique, the power management unit circuit enables a fine-grained control of the photovoltaic cell voltage, which leads to better maximum power point tracking performance and higher energy efficiency. This subsequently results in notable improvement in energy collection and operation time of the device. We note that the IoT devices equipped with the above energy harvesting units can well integrate the systematic and optimized checkpointing methodologies proposed in the Chime framework.

In an earlier work, we have proposed Idetic, a hardware-based checkpointing framework for intermittently powered systems [9]. Our work includes a set of optimization algorithms to locate the checkpoints with the goal of minimizing the cost of storing and retrieving data. The framework targets specific applications where the flow of data within the CDFG is predictable and input-independent. Based on that assumption and during the design time, the application is run for an arbitrary input and the resulting Finite State Machine (FSM) is unrolled to find the checkpoints. Those methods are not applicable for input-dependent applications where each input signal result in a different unrolled FSM. Our other work introduces hardware checkpointing mechanisms for input-dependent algorithms [10]. Such algorithms appear in a much wider range of applications such as computational sensing analysis. This work, however, only focuses on optimized checkpoint placement and activation procedures, and does not offer any charge buffering and migration required for activating the checkpoints.

## 3 Preliminaries

In this section we briefly describe the concept of control data flow graph which is exploited in our checkpointing algorithm. We also discuss Chime'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 [27]. Several HLS tools have emerged that enable automatic synthesis of high-level specification codes such as C/C++ to low level RTL specifications optimized for ASIC or FPGA implementations. In this work, we use Vivado HLS [1] for our high-level synthesis and implementation purposes. Vivado HLS automatically generates CDFG from high-level C/C++ codes.

**Energy harvesting platform:** Chime targets general batteryless devices with intermittent power source. For experimental evaluations, we adopt the energy harvesting model from [15]. 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.

## 4 CHECKPOINTING OVERVIEW

Chime applies the checkpointing method in two phases: design time and real time operation. During the design time, Chime locates the checkpointing spots and automatically embeds the checkpointing circuit for storing data. 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 maximum distance between two consecutive checkpoints is upper-bounded to avoid recomputation due to the power failures. 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. Chime automatically locates the checkpoints and embeds the circuit within the HDL code of the design.

During the real time operation, the available energy source is sensed before each checkpoint operation to decide whether or not to activate the checkpoint (on the already established checkpointing circuits). This avoids unnecessary checkpoints when the power supply can sufficiently feed the system. The global flow of our approach is illustrated in Figure 1.

## 5 CHECKPOINT PLACEMENT: COST QUANTIFICATION

In this section, we explain our algorithms for finding the best locations to insert the checkpoints. First, we begin with a motivational example that shows the importance of checkpoint placement strategy on system's performance. Next, we explain how to find the checkpointing and recomputation energy overhead at different states. In the following section, using the computed costs, we present our 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.

### 5.1 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 show 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 (S1) is 2 energy units. We compare two checkpointing strategies; Strategy 1 inserts the checkpoints at points A1 and A2 and Strategy 2 inserts the checkpoints at points B1 and B2. The figure shows the checkpointing cost at each point, e.g., the cost of checkpointing at A1 is 1 energy unit.

| Failure at state: | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| First strategy's resource loss | 2 | 5 | 12 | 22 | 30 |
| Second strategy's resource loss | 2 | 6 | 9 | 21 | 14 |

TABLE 1: Energy loss is calculated for failures at different states.

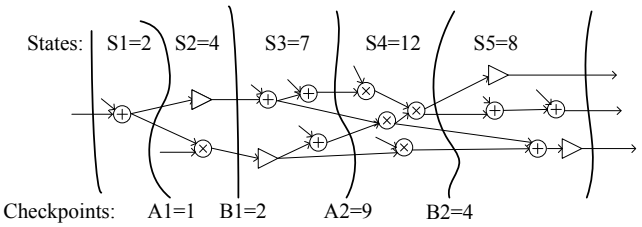


Fig. 2. Different checkpointing strategies affect the performance. Computational cost of the states (S1, S2, ..., S5) are shown. Checkpoints are marked as A1 and A2 (Strategy 1), B1 and B2 (Strategy 2) with numbers next to them indicating their cost. Strategy 2 outperforms Strategy 1.

Now we calculate the energy loss caused by failures at different states. If the failure occurs at the end of state 1 (S1), the loss for both strategies is 2 units since we spend 2 energy units on S1 (which is the cost of S1). If the failure occurs at state 2 (S2), the energy loss for the first and second strategies are 5 and 6 units respectively. In Strategy 1, since A1 is checkpointed, we only lose the cost of S2 which is 4 units. We also spend 1 unit for checkpointing at A1. Thus in total we lose 4+1=5 units. In Strategy 2, for a failure at S2, we lose all resources that are spent at S1 and S2, that is 2+4=6 units. Same loss calculation method can be used to compute the resource loss associated with each strategy for failure at different states. Note that for failures that occur after second checkpointing in each scenario, the previous checkpointing costs should be also considered in the resource loss calculation as we have already spent those extra energy units to checkpoint the circuit before failure occurs. Table 1 shows the resource loss for failures at different states for the two strategies. Assuming that the failures happen with equal probability, Strategy 2 outperforms Strategy 1 since the sum of the losses incurred by Strategy 1 is 71 units while it is 52 units for Strategy 2. The reason behind this is that the total cost of checkpointing at B1 and B2 is less than that of A1 and A2. In addition, checkpoints B1 and B2 are inserted at the end of the states that consume more energy.

### 5.2 Computing cost function

Chime 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.

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)

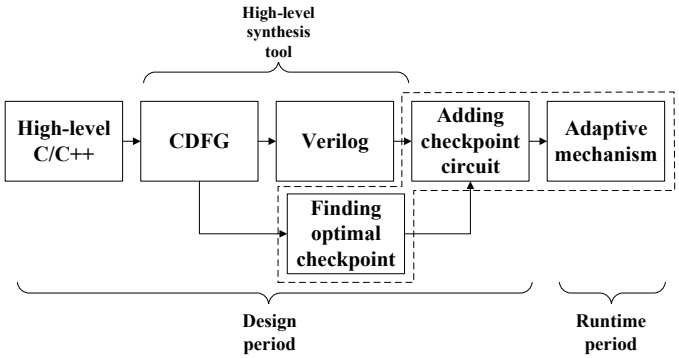


Fig. 1. Global flow of our checkpointing design. Chime applies the checkpointing method in two phases: design time and real time operation. During the design time, Chime locates the checkpointing spots and automatically embeds the checkpointing circuit for storing data. During the real time operation, an adaptive sensing mechanism is applied to decide whether or not to activate the checkpoint.

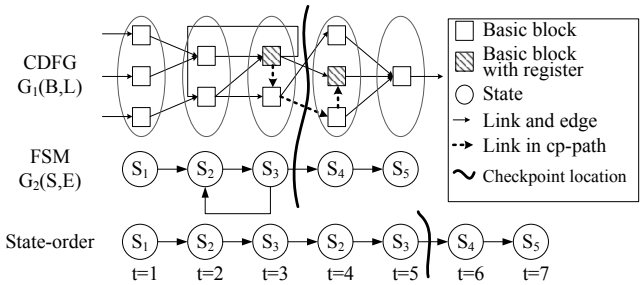


Fig. 3. CDFG, FSM, and State-order of the design. An example checkpoint is inserted between $s_3$ and $s_4$.

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, for recovering it the value of the register $b_n$ should be stored. 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 $\Phi$. For an edge $e_{s_i s_j}$ in $E$, we initially set $\Phi$ as empty (Line 1-2). Next, we add to $\Phi$ 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 $\Phi$ (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 4). For finding the computational energy cost at each state, we performed power simulations using Synopsys Design Compiler.

**Pseudocode-1: Calculate Cost of Checkpointing.**

```
1  for e_{s_i s_j} ∈ E
2     Φ = ∅
3     for s_v : 1 ≤ v ≤ j − 1
4        for s_u : j ≤ u ≤ M
5           for b_n, b_m which b_n ∈ s_v and b_m ∈ s_u
6              if P(b_n, b_m) = true
7                 Φ = Φ ∪ b_n
8              if P(b_m, b_n) = true
9                 Φ = Φ ∪ b_m
10    if j ≤ i
11       for s_v : j ≤ v ≤ i
12          for b_n, b_m which b_n ∈ s_v and b_m ∈ s_j
13             if P(b_n, b_m) = true
14                Φ = Φ ∪ b_n
15    Cost(e_{s_i s_j}) = 0
16    for b ∈ Φ
17       Cost(e_{s_i s_j}) = Cost(e_{s_i s_j}) + width(b)
```

## 6 Checkpoint Placement: Optimization Algorithms

The CDFG and FSM of the design give general information about the flow of data during the computation process. However, they do not provide complete information about runtime behavior of the system. In the following we propose two separate methodologies for inserting the optimal checkpoint locations. The first method targets scenarios where the CDFG is input-independent, i.e., different inputs do not change the computational flow. The second method targets input-dependent CDFGs.

### 6.1 Optimized algorithms for input-independent CDFGs

For finding the optimal checkpoint locations, Chime 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 $C_{OF}^{ind}$** |
|---|---|
| 1 | for $t : 0 \leq t \leq T$ |
| 2 | for $k : 0 \leq t \leq K_{max}$ |
| 3 | $C_{OF}^{ind}(t,k) = +\infty$ |
| 4 | for $t : 0 \leq t \leq D_{max}$ |
| 5 | $C_{OF}^{ind}(t,0) = -C_{CO}(t)$ |
| 6 | for $k : 1 \leq k \leq K_{max}$ |
| 7 | for $t : k \leq t \leq T$ |
| 8 | $C_{OF}^{ind}(t,k) = C_{CP}(t) + \min_{1 \leq i \leq D_{max}}$ |
| 9 | $\{C_{OF}^{ind}(t-i,k-1) - C_{CO}(t) + C_{CO}(t-i)\}$ |

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:** To minimize the checkpointing overhead, it is desirable to insert the checkpoints only before power failures. However, the source unpredictability does not allow us to do so. Based on the energy source capacity and average computation power consumption, Chime adjusts the distance between two consecutive checkpoints. A more conservative approach sets a smaller distance between two checkpoints which ensures the application will be completed in a long run but it introduces more overhead. A larger distance between the checkpoints can be more efficient but may increase the risk of computation loss.

Table 2 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 functions is denoted by $C_{OF}^{ind}(t,k)$ which is the overall energy for completing $k$ checkpoints at the end of the $t^{th}$ state of the state-order (for input-independent CDFGs). 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.

| | |
|---|---|
| $T$ | 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 |

TABLE 2: Defining parameters.

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 (Lines 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 plus the minimum overall energy that is consumed before the $t^{th}$ state (Lines 6-9).

### 6.2 Optimized algorithms for input-dependent CDFGs

For locating the checkpoints in input-dependent CDFGs, additional modifications are required. The goal of our algorithm is to place the checkpoints such that the total energy to execute all the states in the state-order is minimized. Concurrently, our algorithm should ensure completion of the application for different input signals; for the same application (such as FFT-based peak detection) different inputs may result in varying state-orders. The variability of state-order is challenging since during runtime there is no such flexibility to dynamically change the location of checkpoints based on the input data.

To address this challenge, we devise a new corresponding algorithm based on the following key observations: the structure and number of different states in design's FSM is independent of the input. However, based on the input, the number of times that data flows through the loops varies.

Figure 3 shows the execution of an algorithm for one specific input. If the input changes, the overall structure of CDFG and FSM remain the same but different state-orders might be created. For this example, data can flow through the feedback loop between $S_2$ and $S_3$ for a different number of times for each input, resulting in state-orders with various lengths. Another observation is that for many applications such as FFT-based peak detection algorithms, the number of different states in FSM is limited. Thus, it is reasonable to add a checkpointing circuit at the end of each feedback loop. We refer to that location by *loop-end*. For instance, such loop-end state is $S_3$ on Figure 3. That way, if for some input, data iteratively flows through the loop for a large number of times, given the checkpointing circuit, we can save the computational progress. Otherwise, if no such circuit exists, we may never be able to complete the application due to the limited energy storage capacity and the transient source.

Similar to our approach for input-independent checkpointing, we mitigate the source transient behavior by setting a limit on the distance between two consecutive checkpoints. The limit is calculated based on the average power consumption of the computational application and the energy supply capacity.

Pseudocode-3 demonstrates our approach for inserting checkpoints for input-dependent benchmarks. Our assumption is that we have already placed the loop-end checkpoints. Here we run the algorithm to locate additional checkpoints between each two consecutive loop-end states in order to meet the $D_{max}$ criteria. Thus, if two loop-end checkpoints are closer than $D_{max}$ states from each other, no more checkpoints will be added in between them. Here our objective function denoted by $C_{OF}^{dep}(t,n)$ measures the cost of inserting additional checkpoints in between two consecutive loop-ends. Thus, Pseudocode-3 should be executed for finding the checkpoints in between any two subsequent loop-ends whose distance is more than $D_{max}$ states.

First, we set the initial conditions by enforcing a checkpoint at a distance equal or less than $D_{max}$ from the beginning loop-end state (Lines 1-5). To meet the maximum distance constraint, the total number of checkpoints should be at least equal to $\frac{\Delta T_{max}^{LoopEnd}}{D_{max}}$, where $\Delta T_{max}^{LoopEnd}$ is the number of states between the two consecutive loop-ends in FSM. For exploring the effect of different number of

| | **Pseudocode-3: Dynamic Programming for minimizing $C_{OF}^{dep}$** |
|---|---|
| 1 | for $t : 0 \leq t \leq \Delta T_{max}^{LoopEnd}$ |
| 2 | for $n : 0 \leq t \leq N_{max}$ |
| 3 | $C_{OF}^{dep}(t, n) = +\infty$ |
| 4 | for $t : 0 \leq t \leq D_{max}$ |
| 5 | $C_{OF}^{dep}(t, 0) = -C_{CO}(t)$ |
| 6 | for $n : 1 \leq n \leq N_{max}$ |
| 7 | for $t : n \leq t \leq \Delta T_{max}^{LoopEnd}$ |
| 8 | $C_{OF}^{dep}(t, n) = C_{CP}(t) + \min_{1 \leq i \leq D_{max}}$ |
| 9 | $\{C_{OF}^{dep}(t - i, n - 1) - C_{CO}(t) + C_{CO}(t - i)\}$ |

checkpoints on our objective function, we run the algorithm for up to $N_{max}$ number of checkpoints, where $N_{max}$ is equal to $\frac{\Delta T_{max}^{LoopEnd}}{D_{max}} * 2$. The minimum cost for placing the $n^{th}$ checkpoint at the state $t$ is achieved by adding the checkpointing cost at state $t$ and the minimum overall energy cost of inserting the rest of $(n-1)^{th}$ checkpoints before state $t$ (Lines 6-9).

### 6.3 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 real time mechanism that senses the available source energy before each checkpoint. A checkpoint is skipped if the source energy level is greater than the maximum energy consumption between two consecutive checkpoints; in our target applications, this value can be measured offline. 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 and remote energy transfers with limited energy capacities and charging rates.

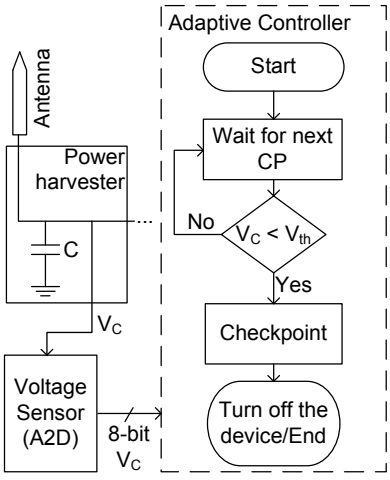


Fig. 4. Adaptive sensing and checkpoint activation mechanism.

## 7 Energy Supply Module for Checkpointing Circuit

The purpose of checkpointing is to preserve the developments in computation in the case of energy cut-offs. To save such developments, it is necessary that we use non-volatile memory. As a result of our efficient checkpoint locating schemes, generally a small amount of data has to be stored on NVM. Thus, the time and energy overhead of writing the checkpointed data onto the memory is quite low (See Section 9). However, the power requirement for writing to NVM can be several times higher than that of the computational IC. Table 3 provides the requirements for writing data onto a contemporary NAND Flash, and PCM. The values show the overhead of writing a page of data. As we discuss in the experiments (Section 9), the average power for our computational ICs is much lower (less than $0.4mW$).

| | Page size (bytes) | Programing current (mA) | Programing voltage (V) | Average power (mW) |
|---|---|---|---|---|
| NAND Flash | 2,112 | 15-30 | 2.7-3.6 | 70.8 |
| PCM | 64 | 0.05-0.10 | 0.9-3.6 | 1.7 |

TABLE 3: Cost of writing checkpointing data to NVM [28], [29].

The high power requirement imposes unwanted effects on the already limited energy source of the IC. For example, it is well-known that batteries suffer from low power densities and exhibit exponentially sharp density reductions in case of high power [30]. Thus, Chime can benefit from a separate low-overhead energy module that is responsible for feeding the checkpointing circuit and transferring the electric charge from the energy source to the CPC.

We propose a customized and low-overhead energy supply module for the CPC. Our design utilizes a set of heterogeneous capacitors. Our approach is based on the following observations. On the one hand, a relatively small capacitor (with a capacity much lower than that of the energy supply of IC) is sufficient to deliver the needed high power to CPC. On the other hand, the charge of a small capacitor disappears (leaks) very quickly during the circuit idle time. Thus, it is necessary to ensure that this capacitor is charged closely before the checkpointing is initiated. Using our knowledge of the checkpointing activation, our design ensures such behavior.

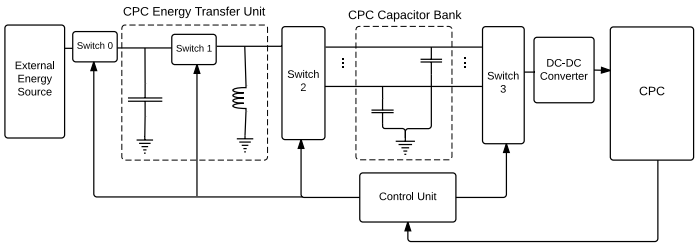


Fig. 5. Schematic depiction of our proposed topology for energy transferring.

Figure 5 illustrates the overview of our proposed energy transferring topology. Here, the large capacitor on the left is considered to be the primary energy supply of the IC.

The control unit is responsible for managing switches in the circuit for efficient energy delivering customization. With the prior knowledge of the load, the control unit estimates the time by when the small capacitors need to be recharged. In our proposed design, while one of the smaller capacitors is connected to CPC, the other small capacitor (*the back-up capacitor*) is recharged by the available charge in the large capacitor.

Another advantage of using small capacitors is that they reach the minimum voltage required for activating the CPC much faster and with less required electric charge. This is because for a given amount of charge ($Q$) a capacitor with a smaller capacitance ($C$) reaches a higher voltage level (since $Q = CV$), and the timing rate of filling to that charge is proportional to $C$. The large capacitor instead incurs lower power leakage in comparison for the same amount of charge.

To efficiently transfer the available charge from the large capacitor to the small ones, we make use of an inductor and two switches which are managed by the control unit. In this topology first, switch 1 connects the large capacitor and the inductor to each other. Once a certain (pre-determined) amount of the available charge (e.g. the amount of charge required to recharge the small back-up capacitor) is transfered to the inductor, switch 1 is disconnected and switch 2 connects the inductor and the back-up capacitor. When the back-up capacitor is recharged it then proceeds to replace the other small capacitor that has already been discharged in the load. By using two alternating small capacitors, we allow checkpointing to be performed smoothly and without interrupt. This procedure continues until the CPC completes its task.

## 8 Checkpointing using JTAG

JTAG is an IEEE standard (1149.1) that is usually used for programming, debugging, or detecting electronic boards manufacturing issues. For circuits that have this built-in facility, one can make use of the available JTAG and our proposed methodologies for checkpointing purposes. In this scenario, since our approach would utilize the circuit built-in facilities, it incurs no area overhead while the time overhead is increased due to the serial nature of the JTAG interface. Figure 6 demonstrates an example CDFG in which the built-in JTAG is used and two locations have been chosen for checkpointing. Note that, in our approach we store the information required to restart the computations from each checkpoint location, however, otherwise all states' information should be stored which is not ideal particularly for scenarios where storage is severely limited. In cases where test compression is used for JTAG circuitry [31], Chime stores the compressed checkpointing responds and reloads the decompress data to resume computations.

For JTAG technologies that enable bypassing registers, the users are provided with a standard interface for checkpointing their circuit. In this case one can use our proposed methodologies and activate JTAG only in the optimally found checkpointing spots. Figure 7 demonstrates an example CDFG in which JTAG has been inserted in two locations for checkpointing purposes. This approach incurs less runtime and energy overhead compared to the conventional JTAG circuitry as it skips the unwanted registers and only passes through the checkpointing register values.

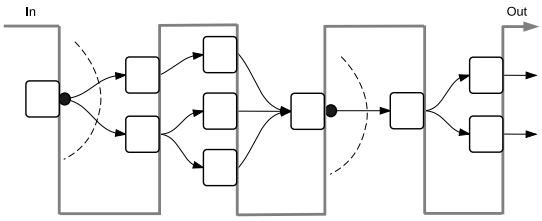


Fig. 6. Checkpointing example using a conventional built-in JTAG. The serial JTAG circuitry passes through all the registers on chip.

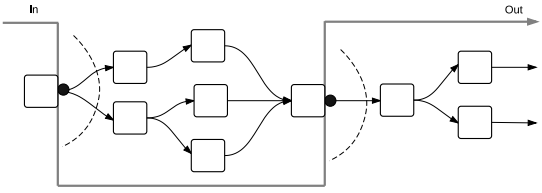


Fig. 7. Checkpointing example using a bypass JTAG. The serial JTAG circuitry bypasses all registers and is only activated at the checkpointing locations.

## 9 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 Chime's applicability and effectiveness.

### 9.1 Evaluation platform

We use Xilinx HLS tool, Vivado HLS, to obtain CDFG and Verilog register-transfer-level (RTL) description from C/C++. Vivado HLS produces CDFG as an intermediate output in standard Extensible Markup Language (XML) format. We implement a C# program to extract CDFG and FSM form the XML file. As discussed in Section 6, we 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. We have also used Synopsis Design Compiler with FreePDK 45nm library [32] to evaluate power consumption and area overhead. We followed the setup shown in Figure 8 for locating and embedding the checkpoints at the RTL to facilitate the integration of the proposed checkpointing solution in the manufacturing flow of the circuits. Note that, Vivado HLS compiles only a subset of standard C/C++ for hardware implementation. Thus, we modify the benchmark source code structure accordingly.

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 operational 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 source capacity is 3.3$\mu$F (except for RSA benchmark which is 10$\mu$F). Thus, the

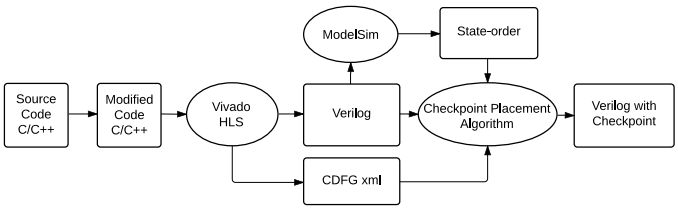


Fig. 8. Depiction of our automated tool for checkpoint placement.

| | 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× |

TABLE 4: NVM properties based on [33], [34].

amount of available energy in the period from $V_{on}$ to $V_{off}$ equals 33.26$\mu$J (100.8$\mu$J 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 and Spin Torque Transfer Memory are two emerging NVM technologies that exhibit better energy and speed performances. The properties of these memories are shown in Table 4.

### 9.2 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 used an oscilloscope (Tektronix, MSO 5054) to capture the capacitor's voltage. 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.

### 9.3 Checkpointing circuit

For an input high-level source code, Vivado HLS 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 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 connection between CPC modules are via relatively small 35-bit buses. 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 includes memory controller circuit and is connected to the NVM.

At the time of checkpointing, the root CPC receives a permission to start checkpointing. Before starting its own checkpointing, the root gives permission to other 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. Checkpointing procedure is completed when the root CPC sends its own data on the memory bus. Figure 9 shows a simple tree structure of CPCs. The numbers in the circles represent order of sending data (lowest number sends first).

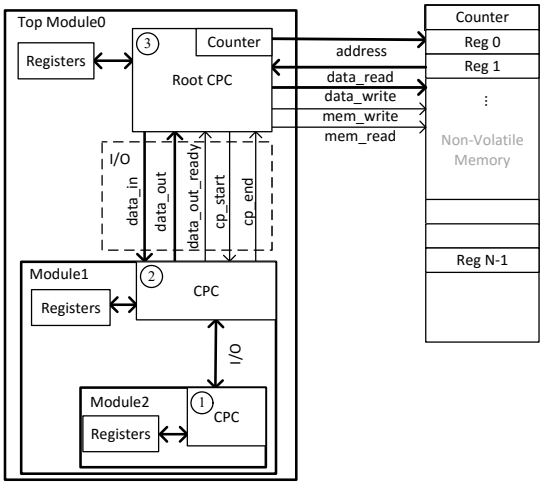


Fig. 9. Checkpointing circuit (CPC) tree structure and depth first order for sending data (lowest number sends first).

### 9.4 Benchmarks

To corroborate the effectiveness of our proposed framework, we evaluate Chime on a set of detection, computational, and security algorithms that are all applicable to medical implant devices. Our benchmarks consist of both input-dependent and input-independent algorithms. Our input-dependent benchmarks include FFT-based peak-detection methods (e.g., in ECG) with varying lengths, and matrix-vector multiplications (e.g., used for sensing analysis). Moreover, our input-independent benchmarks include different cryptographic circuits such as: RSA [35], a public-key cryptography; AES [36], a symmetric-key cryptography; and cryptographic hash functions MD5 [37], SHA1, SHA256, and all five SHA3 round-3 candidates (BLAKE, Grøstl, JH, Keccak, and Skein) [36]. Our RSA benchmark encrypts 1024-bit data under a 1024-bit public key and a constant exponent (65537). AES benchmark encrypts 128KB data under 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. Note that in all of our evaluations each method has been evaluated based on its input-dependency orientation, unless otherwise specified.

**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.

### 9.5 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 10. 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 checkpointing overhead, 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 use of very small number of checkpoints increases the recomputation overhead due to insufficient checkpoints to complete the pertinent computations. The optimal number of checkpoints for MD5 is 4 for our platform.

Figure 11 shows the capacitor voltage behavior over time for completing MD5, and running FFT256. The checkpoint locations and the corresponding computational progresses are shown in the figure. The adaptive mechanism turns off the device after the checkpoint to avoid recomputation and save energy. For instance in the MD5 example, 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.

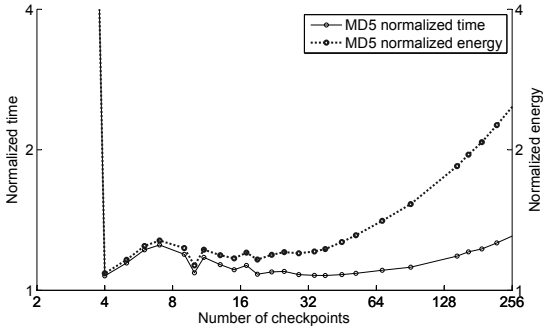


Fig. 10. MD5 normalized energy and time behavior for different number of checkpoints.

We explore the effect of different power source patterns on Chime's performance. We run AES, MD5, SHA256, and FFT64 benchmarks with four synthetic (S1-S4) and four real power traces (S5-S8). Figure 12 reports the corresponding

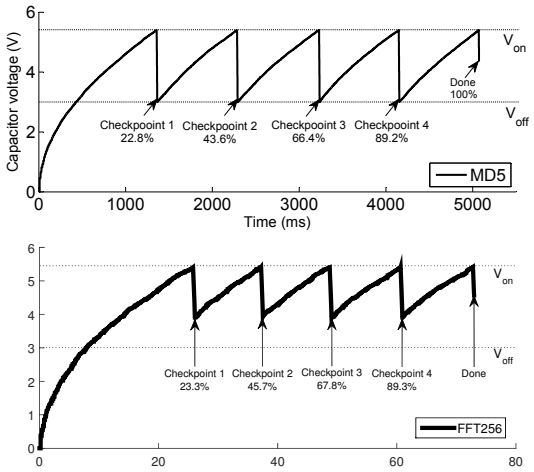


Fig. 11. Checkpoint locations and corresponding progresses for MD5, and FFT256.

normalized energy values. The average power of all the traces are 3.5$\mu$W which is at least 2 orders of magnitude less than average power consumption of the benchmarks. As can be seen on the figure, normalized energy values of all four benchmarks do not considerably change over the synthetic and real power traces. In scenarios, where the rate of energy consumption dominates the rate of energy harvesting, such as our target applications, the traces pattern do not 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.

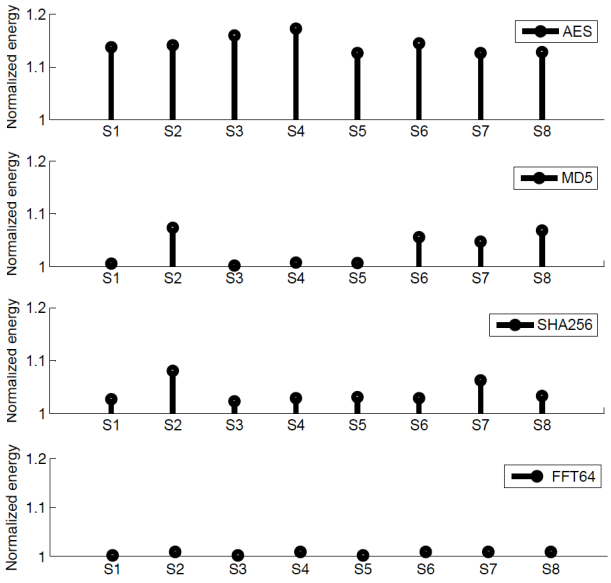


Fig. 12. AES, MD5, SHA256 and FFT64 normalized energy for different power traces with equal average power and various patterns.

To verify the effectiveness of our dynamic programming algorithm, we compare it with the periodic checkpointing approach. Figure 13 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 consump-

tion. 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.

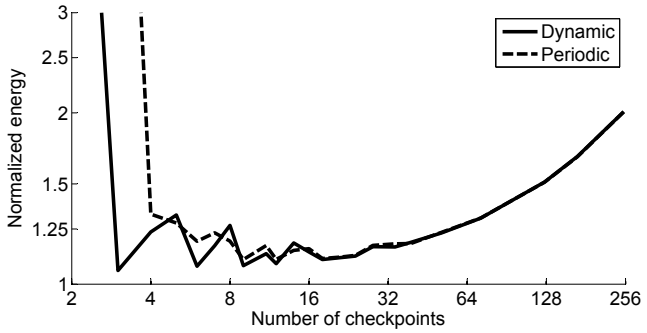


Fig. 13. Performance of dynamic and periodic methods on SHA256.

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 consumption (which is 1.85mW in our experiments). Figure 14 demonstrates 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.

Chime 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 14. In Figure 15, 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 30% 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.

To study the effect of different NVM technologies, we use NAND Flash, a state-of-the-art technology, and two emerging technologies, PCM and SSTM in our simulations. Table 4 shows the characteristic of these memories. We measure time and energy overheads of the benchmarks based on the memory properties, and report the results in Figure 16. In this experiment, all applications have been evaluated based on our input-dependent approach. On the figure, MV300k and MV400k refer to matrix-vector multiplication designs. The figure shows the average results of ten synthetic power traces. The average power of the traces is two order of magnitude less than the power consumption of the benchmarks. The impact of NVM characteristics can be better observed in some applications. However, given the lower cost of Flash and the relative gains, using a more efficient memory may not be justified.

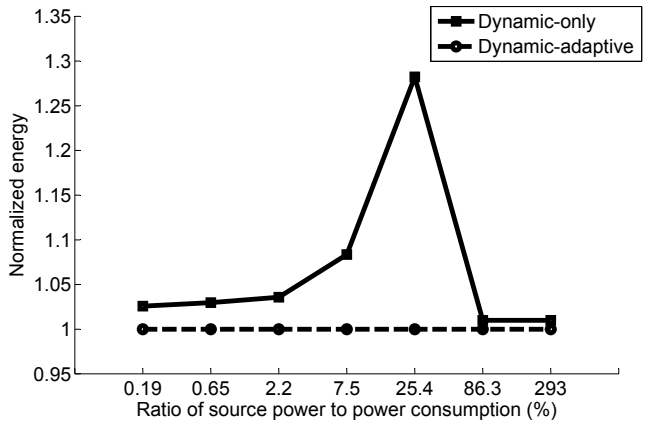


Fig. 14. Comparison of dynamic-only and dynamic-adaptive mechanisms on SHA256 for different mean source powers.

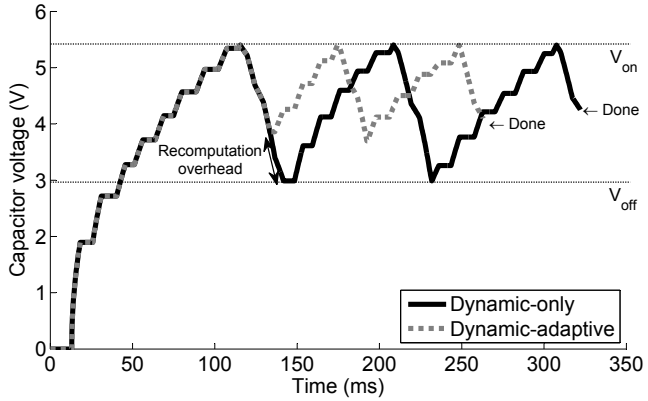


Fig. 15. Simulated capacitor voltage for dynamic-only and dynamic-adaptive methods as SHA256 is running.

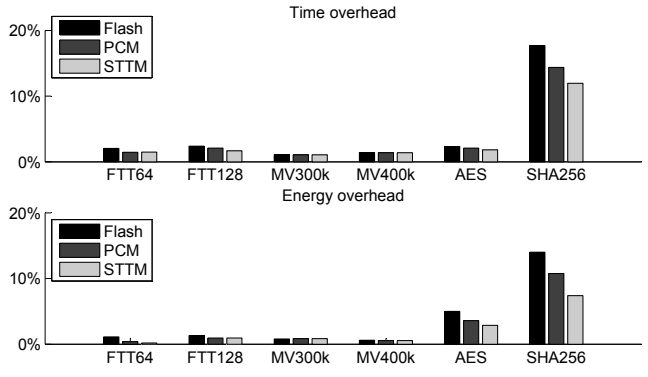


Fig. 16. Time and energy overheads for using PCM, Flash, and STTM.

To further evaluate the effect of different NVM technologies, we measure the corresponding normalized energy and time metrics versus the number of checkpoints, Figure 17. 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 is low, the mem-

ory 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.

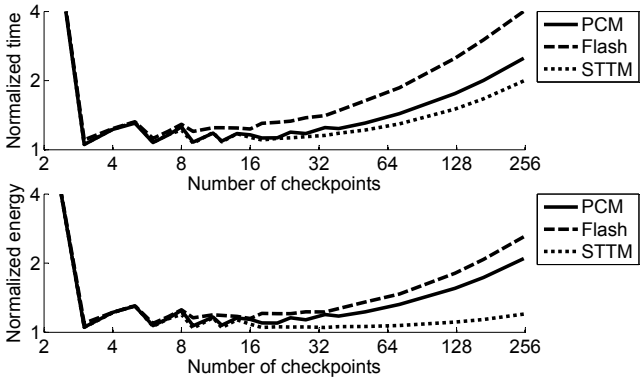


Fig. 17. SHA256 normalized energy and time values for PCM, Flash, and STTM.

Table 5 shows the time overhead for different benchmarks for our JTAG based CPC design discussed in Section 9.3. For our evaluations JTAGs with a speed of $100Mbps$ are used, which is a typical speed for an operation frequency of $915MHz$ (The Moo board working frequency). The memory type is PCM. The capacitor is 3.3$\mu$F (10$\mu$F for RSA). The checkpoints incur an time overhead of less than 1.6%, time overhead of for a conventional JTAG and a timing overhead of less than 0.16% for JTAGs with bypassing capability. The low overhead of our approach experimentally verifies the practicability of Chime in realization of different computing circuits on energy harvesting devices. Table 5 also provides the total number of registers as well as the number of checkpointing registers and the optimal number of checkpoints for each of the benchmarks, that is a direct output of our dynamic checkpointing methodology.

Table 6 shows the time, energy, and area overheads for different benchmarks for our customized CPC architecture discussed in Section 9.3. The memory type is PCM. The capacitor is 3.3$\mu$F (10$\mu$F for RSA). The checkpoints incur an energy overhead of less than 11%, time overhead of less than 16%, and area overhead of less than 9%. The results verify effectiveness and applicability of Chime. As an example consider the RSA benchmark. 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 (10$\mu$F). Thus, our methods enable running a much longer computation on a small capacitor.

Note that for the JTAG based CPC design, we do not provide the energy and area overhead of checkpointing as those overheads are directly affected by the specific implementation model of the JTAG within the circuit. For example, depending on the number of active pins that are assigned for test data in and test data out for various verification purposes, JTAG might incur different timing and energy overhead that are independent of the checkpointing methodology. Our reported timing overhead analysis, however, assumes that a pin that is assigned for checkpointing serially passes the register values through the JTAG circuitry. Serial transfer of register values is the fundamental property of JTAG circuitry design. However, for our custom designed CPC, we are able to estimate the energy, runtime, and area overhead of the checkpointing circuitry. To do so, we implement the nested tree structure proposed in Section 9.3 in Verilog and use the Synopsis Design Compiler with FreePDK library. The total energy overhead of checkpointing is the summation of the energy used by the CPC circuit and that required for writing data onto NVM.

| | Input-dependent | | Input-Independent | | Improvement | |
|---|---|---|---|---|---|---|
| | Time | Energy | Time | Energy | Time | Energy |
| AES | 2.1% | 3.57% | 0.4% | 1.4% | 5.2× | 2.5× |
| SHA256 | 14.4% | 10.8% | 1.8% | 1.8% | 8.0× | 6.0× |

TABLE 7: Comparing the effect of our optimizations for input-dependent versus input-independent checkpointing methodologies.

To experimentally compare the efficiency of our proposed optimizations based upon the input-dependency orientation of the benchmarks, we evaluate both of our approaches on AES and SHA256. Both AES and SHA256 have input-independent CDFGs. The reported results are for our customized CPC architecture. Table 7 shows the normalized time and energy overhead of checkpointing when either of the input-dependent or input-independent approaches are applied to the benchmarks. It can be seen that taking advantage of the input-independent nature of these benchmarks' CDFGs results in much lower checkpointing overhead. For example, for SHA256 benchmark, we achieve 8 folds improvement for time overhead and 6 folds improvement in energy overhead compared to the scenario in which this benchmark has been check-pointed using our general input-dependent approach. This evaluation clearly demonstrates the effectiveness of our proposed framework for checkpointing different methods according to their input-dependency orientations.

## 10 Conclusion

This work proposes a set of methodologies and supported designs for enabling long-running and complex computations on IoT devices with limited and intermittent energy sources. Our approach devises techniques for locating optimal checkpoints that facilitate gradual progresses of computations as the power becomes available. We propose algorithms for benchmarks with input-independent and input-independent control data flow graphs. We provide mechanisms for supplying the checkpointing circuit and adjusting the checkpointing rate in realtime according to the power source variations. We provide automated tools that takes the high-level synthesis design as the input and outputs the Verilog description of the design with embedded checkpoints. Our experiments target a wide range of linear algebra, data transformation, and cryptographic benchmarks. We evaluate the performance for different power source conditions. The results verify the effectiveness of our approach in recomputation cost reduction caused by the power outages. The overhead associated with the

| Benchmarks | Total number of registers | Number of CP registers | Number of CPs | Time overhead bypass JTAG(%) | Time overhead JTAG(%) |
|---|---|---|---|---|---|
| RSA | 92895 | 22742 | 83 | 0.008 | 0.03 |
| AES | 11590 | 822 | 3 | 0.01 | 0.13 |
| MD5 | 13566 | 1096 | 4 | 0.004 | 0.05 |
| SHA1 | 7845 | 274 | 1 | 0.003 | 0.11 |
| SHA256 | 18279 | 822 | 3 | 0.07 | 1.59 |
| BLAKE | 21528 | 274 | 1 | 0.007 | 0.55 |
| Grøstl | 33282 | 822 | 3 | 0.005 | 0.23 |
| JH | 38726 | 3562 | 13 | 0.013 | 0.14 |
| Keccak | 18962 | 1918 | 7 | 0.167 | 1.65 |
| Skein | 14672 | 274 | 1 | 0.002 | 0.15 |
| FFT64 | 1931 | 548 | 2 | 0.003 | 0.01 |
| FFT128 | 2053 | 1644 | 6 | 0.004 | 0.005 |
| MV300k | 383 | 274 | 1 | 0.003 | 0.004 |
| MV400k | 548 | 514 | 2 | 0.003 | 0.003 |

TABLE 5: Statistics of the dynamic checkpointing method for various benchmarks. The runtime overheads correspond to our JTAG based checkpointing circuit.

| Benchmarks | Time overhead(%) | Energy overhead(%) | Area overhead(%) |
|---|---|---|---|
| RSA | 1.1 | 1.1 | 1.2 |
| AES | 0.4 | 1.4 | 4.5 |
| MD5 | 5.0 | 5.7 | 3.2 |
| SHA1 | 0.4 | 0.2 | 2.5 |
| SHA256 | 1.8 | 1.8 | 2.1 |
| BLAKE | 0.2 | 0.6 | 1.5 |
| Grøstl | 8.2 | 9.1 | 3.0 |
| JH | 11.0 | 10.3 | 1.2 |
| Keccak | 15.7 | 10.9 | 1.2 |
| Skein | 0.5 | 0.6 | 5.0 |
| FFT64 | 1.46 | 0.37 | 3.2 |
| FFT128 | 2.08 | 0.93 | 3.2 |
| MV300k | 1.06 | 0.85 | 8.5 |
| MV400k | 1.38 | 0.56 | 8.5 |

TABLE 6: Runtime, energy, and area overheads of the dynamic checkpointing method corresponding to our custom designed checkpointing circuit.

checkpoints was shown to be very low: less than 16% energy overhead, less than 11% time overhead and less than 9% area overhead were reported for our benchmarks.

## 11 Acknowledgement

This research is in parts supported by the Office of Naval Research (ONR) award (grant No. N00014-11-1-0885). and Multidisciplinary University Research Initiative (MURI) award (grant No. FA9550-14-1-0351).

## References

[1] "Vivado hls," 2015. [Online]. Available: http://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2015-1.html

[2] "Jtag tutorial." [Online]. Available: http://www.corelis.com/education/JTAG_Tutorial.htm

[3] 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.

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

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

[6] B. Lucia and B. Ransford, "A simpler, safer programming and execution model for intermittent systems," 2015.

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

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

[9] A. Mirhoseini, E. M. Songhori, and F. Koushanfar, "Idetic: A high-level synthesis approach for enabling long computations on transiently-powered asics," in *Pervasive Computing and Communications (PerCom), 2013 IEEE International Conference on*. IEEE, 2013, pp. 216–224.

[10] ——, "Automated checkpointing for enabling intensive applications on energy harvesting devices," in *Proceedings of the International Symposium on Low Power Electronics and Design*. IEEE Press, 2013, pp. 27–32.

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

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

[13] 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.

[14] 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.
[15] 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.
[16] A. Chandrakasan, D. Daly, J. Kwong, and Y. Ramadass, "Next generation micro-power systems," in *VLSI Circuits*, 2008, pp. 2 –5.
[17] 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.
[18] B. Calhoun, A. Wang, and A. Chandrakasan, "Modeling and sizing for minimum energy operation in subthreshold circuits," *SSC*, pp. 1778 –1786, 2005.
[19] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *LCN*, 2004, pp. 455 –462.
[20] 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.
[21] M. Buettner, B. Greenstein, D. Wetherall, and J. Smith, "Revisiting smart dust with RFID sensor networks," 2008.
[22] A. Orailoglu and R. Karri, "Coactive scheduling and checkpoint determination during high level synthesis of self-recovering microarchitectures," *TVLSI*, pp. 304 –311, 1994.
[23] Y. Liu, Y. Wang, H. Jia, S. Su, J. Wen, W. Zhang, L. Zhang, and H. Yang, "Demo abstract: An energy harvesting nonvolatile sensor node and its application to distributed moving object detection," in *Information Processing in Sensor Networks (IPSN)*, 2012, pp. 149–150.
[24] 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 *ESSCIRC*, 2012, pp. 149–152.
[25] C. Wang, N. Chang, Y. Kim, S. Park, Y. Liu, H. G. Lee, R. Luo, and H. Yang, "Storage-less and converter-less maximum power point tracking of photovoltaic cells for a nonvolatile microprocessor," in *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*, 2014, pp. 379–384.
[26] H. G. Lee and N. Chang, "Powering the iot: Storage-less and converter-less energy harvesting," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, 2015, pp. 124–129.
[27] 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.
[28] "Micron, nand flash memory datasheet," 2015. [Online]. Available: http://www.micron.com/parts/nand-flash/mass-storage/mt29f2g08aabwp
[29] "Micron, p8p parallel phase change memory (pcm) datasheet," 2015. [Online]. Available: http://www.micron.com/~/media/Documents/Products/Data%20Sheet/PCM/p8p_parallel_pcm_ds.pdf
[30] A. Mirhoseini and F. Koushanfar, "Learning to manage combined energy supply systems," in *Low Power Electronics and Design (ISLPED) 2011 International Symposium on*, Aug 2011, pp. 229–234.
[31] "Synopsys dftmax ultra," 2016. [Online]. Available: https://www.synopsys.com/Tools/Implementation/RTLSynthesis/Test/Pages/dftmax-ultra-ds.aspx
[32] 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.
[33] 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.
[34] S. Chen, P. Gibbons, and S. Nath, "Rethinking database algorithms for phase change memory," in *CIDR*, 2011, pp. 1 –11.
[35] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *CACM*, pp. 120 –126, 1978.
[36] N. I. of Standards and Technology. (2012) Cryptographic technology. [Online]. Available: http://csrc.nist.gov/groups/ST/index.html
[37] R. Rivest, "The md5 message-digest algorithm," *RFC 1321*, 1992.

**A** zalia Mirhoseini is a postdoctoral researcher in the department of Electrical and Computer Engineering (ECE) at Rice University. She received her Ph.D. from Rice University where she worked on algorithms and architectures for performance efficient data analytics. Ms. Mirhoseini is the recipient of National Gold Medal in Iran Mathematics Olympiad (2004), Microsoft Women Graduate Student Scholarship (2010), IBM Ph.D. Student Scholarship (2012), and Schlumberger Ph.D. Student Fellowship (2013). Her dissertation received the best 2015 Ph.D. thesis award at Rice University's ECE department.

**B** ita Darvish Rouhani received her B.Sc. in Electrical Engineering from Sharif University of Technology in 2013 and her M.Sc. in Electrical and Computer Engineering from Rice University in 2015. She is currently a Ph.D. student at the Department of Electrical and Computer Engineering at University of California, San Diego. Her research interests include algorithms, low-power computing, distributed optimization, big data analysis, and deep learning. She is a student member of the IEEE.

**E** brahim M. Songhori received the B.Sc. degree in Computer Engineering from University of Tehran, Iran in 2011 and the M.Sc. degree from Rice University, Houston, TX in 2014. He is currently working toward the PhD degree in the Department of Electrical and Computer Engineering, Rice University under supervision of Professor Farinaz Koushanfar (University of California, San Diego). His current research interests include Computer Architecture, Secure Multi-party Computation, Embedded System Design, FPGA and ASIC design, and High Performance Computing. He is a student member of the IEEE.

**F** arinaz Koushanfar is a professor and Henry Booker Faculty Scholar of Electrical and Computer Engineering (ECE) in University of California, San Diego (UCSD) where she is directing the Adaptive Computing and Embedded Systems (ACES) Lab. Before joining the faculty at UCSD, she was a professor of Electrical and Computer Engineering at Rice University. Professor Koushanfar received her Ph.D. in Electrical Engineering and Computer Science and M.A. in Statistics, both from UC Berkeley in 2005. Her M.S. degree is from UCLA and her B.S. from Sharif University of Technology, both in Electrical Engineering. Koushanfars research interests include embedded and cyber-physical systems design, embedded systems security, design automation (DA), and in particular DA of domain-specific/mobile computing and machine learning applications. Professor Koushanfar serves as an associate partner of the Intel Collaborative Research Institute for Secure Computing to aid developing solutions for the next generation of embedded secure devices. Dr. Koushanfar is the founder of Women ExCEl and a co-founder of the Trust-Hub. She has received a number of awards and honors for her research, mentorship and teaching, including the Presidential Early Career Award for Scientists and Engineers (PECASE) from President Obama, the ACM SIGDA Outstanding New Faculty Award, MIT TR-35, and Young Faculty/CAREER Awards from NSF, DARPA, ONR and ARO.