PaCC: A Parallel Compare and Compress Codec for Area Reduction in Nonvolatile Processors

Yiqun Wang, *Student Member, IEEE*, Yongpan Liu, *Member, IEEE*, Shuangchen Li, Xiao Sheng, Daming Zhang, Mei-Fang Chiang, Baiko Sai, Xiaobo Sharon Hu, *Senior Member, IEEE*, and Huazhong Yang, *Senior Member, IEEE*

***Abstract*— Nonvolatile (NV) processors have attracted much attention in recent years due to their zero standby power, resilience to power failures, and instant-on feature. One design challenge of NV processors is the excess area needed by NV registers. This paper introduces a parallel compare and compress (PaCC) architecture to reduce such excess area. A key component of the PaCC architecture is a new codec which effectively balances area and performance. In addition, the PaCC architecture includes a configurable state table to support reference vector selection for different applications. With the proposed vector selection algorithm, the PaCC architecture can outperform other vector selection approaches by over 59% in terms of reduction in the number of NV registers. The proposed architecture has been fully realized at the circuit level and synthesized for the Rohm's 0.13-μm ferroelectric-CMOS hybrid process. Results demonstrate that the design can reduce the number of NV registers by 70%–80% with less than 1% overflow possibility, which leads to up to 30% processor area saving. The overall approach is applicable to any NV processor design regardless of the NV material used.**

***Index Terms*— Area efficiency, data compression, nonvolatile processor.**

I. Introduction

CONVENTIONAL processors keep their states on the CMOS capacitors in volatile storage elements, such as flip-flops, registers, and static random access memories (SRAMs). The states are lost when the power is turned off, because the charge on the capacitors quickly drains without the power supply. Some traditional processors use nonvolatile (NV) memories to back up the states. Some exascale computing systems prefer to use off-chip phase-change random access memory (PCRAM) for checkpointing applications [1], while self-powered embedded systems use on-chip ferroelectric random access memory (FeRAM) to prevent data loss during power failures [2]. However, since they adopt a centralized storage system, the data transfer between a volatile processor and secondary NV storages causes long delays, large switching energy consumption, and vulnerability to power failure. Therefore, efficient alternatives are needed to remove such limitations.

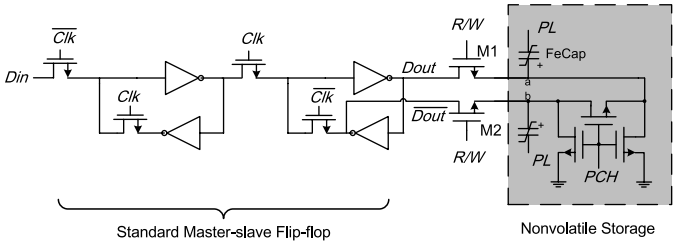


Fig. 1. Example of NVFF realized with ferroelectric capacitors [16].

One promising alternative is NV processors [3]–[6]. Different from conventional processors with secondary NV memories, the NV processors use distributed NV storages at the register level (shown in Fig. 1). The NV flip-flop acts as a standard D flip-flop in normal operations, whereas it stores the present state into the local NV storage in backup operations. Therefore, the NV processor can back up all data in parallel and reduce the backup time and energy by 2–3 orders of magnitude compared to conventional processors [4]. Another alternative is the logic-in-memory structure used in a full adder which retains the state locally [7]. However, its endurance and performance may suffer from the cycle-by-cycle writing operations to magnetic tunnel junctions (MTJs). In general, our proposed NV processor is shown to have the following advantages.

1) *Zero Standby Power:* An NV processor can retain its state when the power is off.
2) *Instant On and Off:* An NV processor can resume its work within only 3 μs from the stalled point [4], while a volatile one needs several milliseconds.
3) *High Resilience to Power Failures:* An NV processor can work reliably under the environments with frequent power interrupts, such as energy harvesting and wireless powered applications [3].
4) *Fine-Grained Power Management Supported [6]:* An NV processor can be shut down whenever possible because of the ultralow energy and fast recovery characteristics.

To implement an NV processor, appropriate NV memory technologies are needed for flip-flops and registers. Flash is

Manuscript received October 10, 2012; revised June 12, 2013; accepted July 25, 2013. Date of publication August 23, 2013; date of current version June 23, 2014. This work was supported in part by the High-Tech Research and Development (863) Program under Contract 2013AA013201, in part by the National Natural Science Foundation of China under Grants 61204032 and 61271269, in part by the National Science and Technology Major Project under Contract 2010ZX03006-003-01, and in part by the International Cooperation from ROHM Inc.

Y. Wang, Y. Liu, S. Li, X. Sheng, D. Zhang, and H. Yang is with the Department of Electronic Engineering, Tsinghua University, Beijing 100084, China (e-mail: ypliu@tsinghua.edu.cn; yanghz@tsinghua.edu.cn; wangyq05@mails.tsinghua.edu.cn; lisc07@mails.tsinghua.edu.cn; zdm10@mails.tsinghua.edu.cn; shengxiao08@mails.tsinghua.edu.cn).

M.-F. Chiang and B. Sai are with LSI HD, Rohm Co., Ltd., Kyoto 6730, Japan (e-mail: meifang.chiang@dsn.rohm.co.jp; baiko.sai@dsn.rohm.co.jp).

X. S. Hu is with the Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46656 USA (e-mail: shu@nd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2013.2275740

a mature high-density NV memory in commercial microcontrollers [8], [9]. However, it has drawbacks such as low endurance, slow writing speed, block erasing pattern, and high mask cost as distributed NV registers. PCRAM has the potential to replace DRAM as main memories [10]. However, its asymmetric reading/writing characteristics and limited lifetime hinders its application as flip-flops and registers. Among the existing NV memories [11], FeRAM and magnetic random access memory (MRAM) emerge as the most promising candidates for NV processors because of their nearly unlimited operation cycles, ultrashort access time, and easy integration to CMOS technology.

Many authors have investigated incorporating FeRAMs and MRAMs into integrated circuits. Zhao *et al*. [12] employed MTJ-based flip-flops in field-programmable gate arrays to achieve rapid start-up time. Sakimura *et al*. [13] developed a magnetic flip-flop library for systems-on-a-chip design. Guo *et al*. [14] conducted an architectural analysis of an STT-MRAM-based processor. Zwerg *et al*. [2] embedded an FeRAM into a microcontroller for better tolerance to power failures. Rohm [15] developed a lifetime-enhanced NV register by adding a ferroelectric capacitor to a standard register.

One challenge faced by all integrated circuits containing NV registers is the significant area overhead. None of the existing works touches this challenge. To see the severity of the area challenge, we examine some existing NV flip-flop (NVFF) designs. In [16], the authors proposed a ferroelectric NVFF which is 4–5× larger than a regular CMOS flip-flop due to its hybrid structure. In [3], floating-gate transistor-based hybrid registers were used to build an NV processor. Their results showed that the NVFFs incurred over 20% chip and 40% memory area overheads. In [13], the authors observed over 40% area increase in a low-pass digital filter by replacing traditional registers with magnetic NVFFs.

In this paper, we introduce a comprehensive solution, referred to as parallel compare and compression (PaCC), to reduce the area overhead due to NVFFs while minimizing performance penalties. The goal of PaCC design is for it to be fast and area-efficient to avoid additional overhead. Furthermore, it should be effective for any programs executed on an NV processor. The PaCC design is built on the key observation that processor states have a significant amount of redundancy over time [4]. Specifically, we make the following contributions.

1) We propose the PaCC architecture to reduce the number of the bits to be stored in the NVFFs, and hence the number and area of NV registers. The architecture adopts a compare and compress strategy to improve the compression ratio.
2) We develop a compression codec based on a parallel run-length encoding/decoding scheme, which can achieve over 5–8× speedup over the conventional serial run-length codec. An area-efficient two-stage shifting network is designed to minimize the codec's area, which reduces the area of the original barrel shifting network by 8–10×.
3) We introduce a configurable state table structure in PaCC to store the reference vectors used for comparison. Given a specific application, we formulate the reference vector selection problem as an optimization problem and develop heuristic algorithms to solve it. For multiple applications, we propose a metric for selecting a proper vector set.

TABLE I
AREA CHALLENGES UNDER DIFFERENT NONVOLATILE APPROACHES

Approach	Area of NVFF in DFFs	Area overhead of Entire Chip
Floating Gate [3]	1.4x	19%
Magnetic RAM [13]	2x	40%
FeRAM [4]	$4-5$x	90%

The reminder of this paper is organized as follows. Section II discusses the area challenges and proposes the PaCC architecture. We introduce the hardware design of PaCC in Section III and present the related control and optimization methods for PaCC in Section IV. We discuss an actual NV processor implementation in Section V and evaluate the proposed PaCC in this processor in Section VI. Section VII concludes this paper.

II. OVERVIEW

This section first uses the actual implementation of NVFF in [16] to demonstrate the area impact of NVFFs based on the hybrid structure proposed in [16]. We then discuss the redundancy in processor states over time. Finally, we introduce the PaCC architecture for reducing the redundant states and bring out elaborate design challenges.

A. Area Impact of NVFFs

An NV processor requires NVFFs to have minimal performance loss during normal operation and low transfer overhead for data backup and restoration. The hybrid NVFF shown in Fig. 1 is such a device. It consists of a standard CMOS flip-flop and an NV storage circuits. In the normal operation, the switching transistors $M1$ and $M2$ are open and the NV storage is isolated. The hybrid NVFF works as a master–salve flip-flop, which has the virtues of short access time, unlimited writing times, and small writing power. In the NV mode, the state in the standard flip-flop is written into the NV storage. $M1$ and $M2$ are shorted and the clock Clk is gated to keep the state unchanged during the backup process. In the recovery process, the data moves in the opposite direction. Therefore, the hybrid NVFFs provide both good performance and long lifetime.

However, hybrid NVFFs lead to significant area increase because they contain NV storage besides standard flip-flops. The NV storage, such as ferroelectric capacitors (see Fig. 1), MTJs, or floating-gate transistors, usually occupies a very large area. Table I shows the area overhead of a single NVFF and an NV processor employing different NV techniques. It shows that the area overhead in different NV technologies are all nontrivial. Note that the area overhead does not consider the

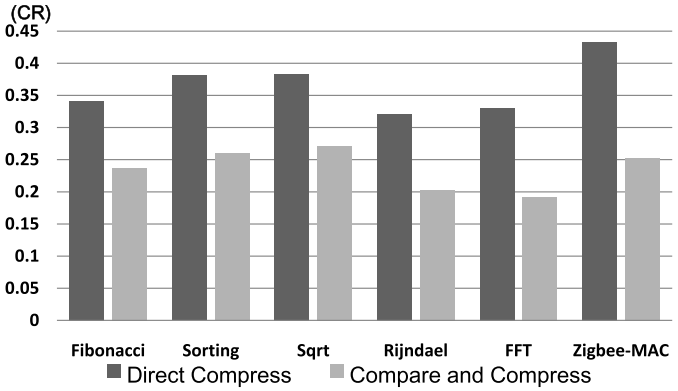


Fig. 2. Compression ratio after conventional RLE compression with or without comparing.

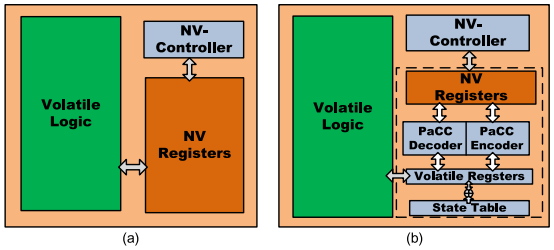


Fig. 3. Conventional architecture versus PaCC architecture. (a) Conventional architecture for nonvolatile processor. (b) Parallel PaCC architecture and nonvolatile processor.

variation and reliability issues, which may further increase the chip area. For example, in the ferroelectric processor, the ferroelectric capacitors have a sandwich structure with a ferroelectric film layer between two metal layers. To reduce error rates in read and write operations to a low level, the ferroelectric film should be large enough. Measurements show that about $5\times$ the original register area is appropriate in a commercial 0.13-μm process. Recently, it was pointed out in [17] that NV memory demonstrates statistical writing and reading behaviors and requires extra error correction units to improve stability.

To quantify the area overhead introduced by NVFFs in an NV processor, we consider an NVFF being α times larger than the original flip-flop and NVFFs occupying β $(0 < \beta < 1)$ of the total chip area. The area overhead S_{ov} is equal to $\beta \times (\alpha - 1)$ when using NVFF instead of a standard flip-flop. For the fabricated processor presented in [4], β is equal to 20% and α is near 5, so $S_{\text{ov}} = 80\%$. Therefore, efficient design techniques are needed to alleviate such a huge impact on the chip area.

B. Redundancy in Processor States

As most of the area overhead comes from the NVFFs, using fewer NVFFs can effectively reduce the chip area. To realize this, we analyze the characteristics of the processor states. Let $v_i = 0/1$ represent the value of the ith flip-flop in a processor at a given time. We use $\mathbf{V} = (v_1, v_2, \ldots, v_n)$ to denote the processor state at that point. Simulations show that over 80% of the v_is are unchanged during a program execution. If we construct a reference vector $\mathbf{V}_{\text{ref}}$ such that each $v_i \in \mathbf{V}_{\text{ref}}$ equals the most common value for the corresponding flip-flop, a differential vector $\mathbf{V}_{\text{diff}} = \mathbf{V} \oplus \mathbf{V}_{\text{ref}}$ can be obtained for a state vector $\mathbf{V}$. As $\mathbf{V}_{\text{diff}}$ may consist of many consecutive zeros, we can compress $\mathbf{V}_{\text{diff}}$ to a much shorter vector $\mathbf{V}^c_{\text{diff}}$ and use fewer NVFFs to store it. We define the compression ratio (CR) as

$$\text{CR} = \frac{|\mathbf{V}^c_*|}{|\mathbf{V}_*|} \tag{1}$$

where $\mathbf{V}_*$ is any original vector and $\mathbf{V}^c_*$ is the compressed version. Fig. 2 shows the CRs for different benchmarks when compressing $\mathbf{V}$ directly and $\mathbf{V}_{\text{diff}}$ with conventional run-length encoding (RLE). As we can see, $\mathbf{V}_{\text{diff}}$ provides over 40% gain to the CR. We will exploit the idea of compressing $\mathbf{V}_{\text{diff}}$ to reduce the NV processor chip area.

C. PaCC Architecture

A straight forward way to implement an NV processor is to simply augment a volatile processor with NV registers and an NV controller [see Fig. 3(a)]. Such a processor stores the system state $\mathbf{V}$ without compression, and the NV registers increase the area significantly. To reduce the area overhead, we propose the PaCC architecture for an NV processor [see Fig. 3(b)]. The PaCC architecture consists of volatile registers which stores the current system state $\mathbf{V}$, a state table, a compression codec (divided into a PaCC encoder and a PaCC decoder), and a small set of NV registers. The state table is used to store $\mathbf{V}_{\text{ref}}$; the compression codec is used to make conversion between $\mathbf{V}_{\text{diff}}$ and $\mathbf{V}^c_{\text{diff}}$; and the comparison is done by the bitwise XORs. Though the volatile registers, the state table, and compression codec may increase the area, the significant reduction in the number of NV registers leads to a much smaller overall chip area.

The operation of PaCC can be partitioned into the encoding procedure and the decoding one. The encoding procedure accomplishes the following:

1) halts the clock to maintain the state vector $\mathbf{V}$ when a power interruption is detected;
2) selects a reference vector $\mathbf{V}_{\text{ref}}$ from the state table, which can generate the most consecutive zeros after comparison;
3) calculates $\mathbf{V}_{\text{diff}}$ by XORing $\mathbf{V}$ and $\mathbf{V}_{\text{ref}}$;
4) compresses $\mathbf{V}_{\text{diff}}$ using the PaCC encoder to get $\mathbf{V}^c_{\text{diff}}$;
5) stores $\mathbf{V}^c_{\text{diff}}$ into the nonvolatile storages.

The decoding procedure works in the opposite direction.

The design of PaCC faces a number of challenges. First, the selection of the compression algorithm must balance its impact on CR and its own area. Complex compression algorithms tend to achieve a low CR, and hence a smaller number of NVFFs, but their nontrivial area overheads may offset the area savings from the NVFFs. Second, the compression operation may compromise the instant on and off features in an NV processor, which requires techniques such as parallelization to speed up the operation. Third, an error-tolerant mechanism should be provided to handle potential overflow when NVFFs

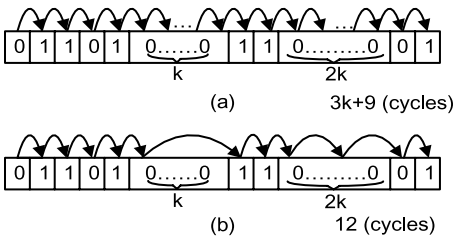


Fig. 4. Example of k-bit parallel RLE. For k-bit 0s or 1s, parallel observation skips all k bits in one cycle. (a) Conventional serial flow. (b) Flow with parallel observation.

cannot hold $\mathbf{V}_{\text{diff}}^{c}$ due to workload variations. Fourth, since the selection of the "best" $\mathbf{V}_{\text{ref}}$ requires high time complexity, a fast heuristic algorithm is needed to find the proper reference vector to minimize CR. Last but not least, as the optimal reference vector values may differ for different applications, we need a technique to select reference vectors for multiple applications while keeping the state stable and small enough. The next two sections detail our approach to tackle these challenges.

III. PACC DESIGN

As discussed in the previous section, the PaCC design includes two main parts: the PaCC codec and the state table. In the codec design, we concentrate on the compression algorithm and its hardware implementation. In the state table design, we concentrate on an efficient architecture supporting multiple applications. In this section, we discuss them in detail.

A. PaCC Codec Design

The PaCC codec design includes two stages. First, a compression algorithm with superior compression ratio should be designed and its hardware feasibility and processing speed should be taken into consideration as well. Second, the hardware design of the codec should be implemented. In the hardware design, we should strive to reduce its area overhead.

1) Customized Compression Algorithm: A proper compression algorithm should have the following features: 1) it should be lossless: the system state should be precisely recovered in the decoding procedure and 2) it should be efficient to implement as hardware: i.e., the area overhead introduced by the hardware realization of the algorithm should be as small as possible. One widely used lossless encoding algorithm is Huffman coding [18], which is optimal as it provides a uniquely decodable entropy code. However, Huffman coding requires a predefined probability distribution of each symbol, which is unobtainable for $\mathbf{V}_{\text{diff}}$. Some efficient binary compression algorithms are used in memory data compression, such as the LZRW [19] series, which achieve good compression speed and ratio. However, those algorithms are software-based and require significant additional hardware to store code the dictionary.

RLE is a lossless algorithm and can be easily implemented in hardware. Furthermore, RLE encoding does not need *a priori* assessment of symbol probability distribution and code mapping. Above all, it can result in impressive compression ratios for consecutive 0/1 sequences, which is the main feature of $\mathbf{V}_{\text{diff}}$. Therefore, we choose RLE encoding as the compression algorithm.

The shortcoming of the traditional RLE is that it processes bit streams serially, which incurs a large time overhead. Trein *et al*. [20] proposed a parallel input-based RLE architecture. However, their structure targets at image applications and considers the encoder only. By extending their parallel input principle to binary sequence, we introduce a parallel observation mechanism into the traditional RLE. Instead of examining a bit stream bit by bit, our parallel RLE (PRLE) observes k bits in parallel. If they are all 0 or 1, all the k bits are bypassed in one clock cycle. Fig. 4 shows an example of compressing a $(3k + 9)$-bit vector. By bypassing k zeros at a time [see Fig. 4(b)], PRLE only consumes 12 clock cycles, whereas the conventional RLE consumes $3k + 9$ clock cycles. We call k the observation window width (OWW) of PRLE. Obviously, the value of k affects the compression speed by impacting the bypass opportunity. We can find an appropriate k for each input bit stream, and its determination is discussed in Section VI.

Another shortcoming of traditional RLE is that it cannot provide efficient compression when meeting short 0/1 chains. Some adaptive length coding algorithms can improve the compression efficiency of short chains by dynamically adjusting the symbol length. However, their challenges come from two aspects: 1) the improvement of the adaptive codec is limited because of the infrequent appearance of such short 0/1 chains and 2) the adaptive codec needs more complex control circuits, which may lead to large area overheads. In order to deal with the compression inefficiency on short 0/1 chains with small hardware overhead, we simply adopt a threshold value L_{th} as in [21]. We only encode the 0/1 chains longer than L_{th} bits because short chains cannot be compressed by our PRLE.

Algorithm 1 presents the details of PRLE. The input of the algorithm is the differential vector $\mathbf{V}_{\text{diff}}$, the OWW k, and the threshold L_{th}. The output is the compression result $\mathbf{V}_{\text{diff}}^{c}$. Variable $\mathbf{S}_{\text{uni}}$ denotes a uniform sequence of all 0s or 1s, and $\mathbf{S}_{\text{non}}$ denotes a nonuniform sequence such as $\{0100101\ldots\}$ which does not contain any 0/1 chains longer than L_{th} bits.

After the initialization (Line 1), the main body of the algorithm is a while loop (Line 2) which checks the end of input vector. We first append the following 0/1 chain to $\mathbf{S}_{\text{uni}}$ as the temporary uniform sequence. In this step, we execute the parallel observation to check whether the following k bits or the remaining bits are all 0s or 1s. If so, we append all of them to $\mathbf{S}_{\text{uni}}$ to bypass them in one step (Lines 4–6). Otherwise, we only add the current bit to $\mathbf{S}_{\text{uni}}$ (Lines 7–9). Subsequently, if a 0/1 transition is detected (Line 10), we decide how to process $\mathbf{S}_{\text{uni}}$. We check whether the length of $\mathbf{S}_{\text{uni}}$ exceeds L_{th}. If not, which means that the current $\mathbf{S}_{\text{uni}}$ actually belongs to a nonuniform sequence, we append $\mathbf{S}_{\text{uni}}$ to $\mathbf{S}_{\text{non}}$ and clear $\mathbf{S}_{\text{uni}}$ (Lines 11–13). If the length of $\mathbf{S}_{\text{uni}}$ is larger than L_{th}, we first call function Process_ShortSEQ to encode $\mathbf{S}_{\text{non}}$ and Process_LongSEQ to encode $\mathbf{S}_{\text{uni}}$, then reinitialize $\mathbf{S}_{\text{non}}$ and $\mathbf{S}_{\text{uni}}$ (Lines 16–20). If reaching the end of $\mathbf{V}_{\text{diff}}$ ($s = n + 1$), we process $\mathbf{S}_{\text{uni}}$ as above (Lines 11–13, 16–20).

Algorithm 1 Threshold-Based Parallel RLE Algorithm

Input: $\mathbf{V}_{diff} = \{a_1, a_2, ..., a_s, ..., a_n\}, L_{th}, k$
Output: $\mathbf{V}^c_{diff}$
Variables: $\mathbf{S}_{uni}$, $\mathbf{S}_{non}$, s

1 *Initialization:* $\mathbf{V}^c_{diff} = \phi$, $\mathbf{S}_{uni} = \phi$, $\mathbf{S}_{non} = \phi$, $s = 1$;
2 **while** $s \leq n$ **do**
3 $w = \min\{s + k - 1, n\}$;
4 **if** $\{a_s, ..., a_w\} == \{00...0\} or \{11...1\}$ **then**
5 $\{a_s, ..., a_w\} \rightarrow \mathbf{S}_{uni}$;
6 $s = w + 1$;
7 **else**
8 $a_s \rightarrow \mathbf{S}_{uni}$;
9 $s = s + 1$;
10 **if** $(a_s \neq a_{s+1}$ && $s \leq n) \| (s == n + 1)$ **then**
11 **if** $|\mathbf{S}_{uni}| <= L_{th}$ **then**
12 $\mathbf{S}_{uni} \rightarrow \mathbf{S}_{non}$;
13 $\mathbf{S}_{uni} = \phi$;
14 **if** $s == n + 1$ **then**
15 $Process\_ShortSEQ(\mathbf{S}_{non}) \rightarrow \mathbf{V}^c_{diff}$;
16 **else**
17 $Process\_ShortSEQ(\mathbf{S}_{non}) \rightarrow \mathbf{V}^c_{diff}$;
18 $Process\_LongSEQ(\mathbf{S}_{uni}) \rightarrow \mathbf{V}^c_{diff}$;
19 $\mathbf{S}_{non} = \phi$;
20 $\mathbf{S}_{uni} = \phi$;

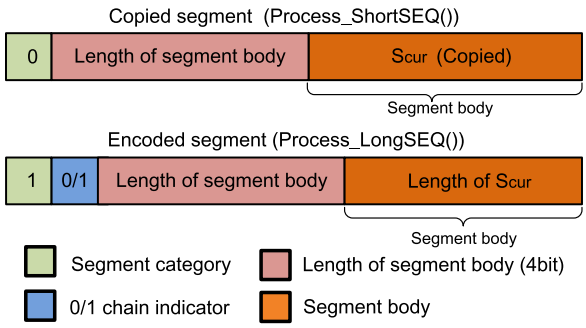


Fig. 5. Encoded and copied segment structure. The copied segment is the format used by chains with mixed 0s and 1s of lengths shorter than L_{th}. The encoded segment is used by the chains with lengths longer than L_{th}.

Especially, we call function Process_ShortSEQ to encode the remaining $\mathbf{S}_{\text{non}}$ (Lines 14 and 15) when $|\mathbf{S}_{\text{uni}}| <= L_{\text{th}}$ occurs. The results of the functions Process_ShortSEQ and Process_ShortSEQ follow the format shown in Fig. 5. We call them copied segment and encoded segment, respectively.

In Algorithm 1, the two input values, L_{th} and k, are critical to the compression ratio and speed. They can be reconfigured based on the actual applications. Their appropriate values are located in small ranges and we will discuss their choices in Section VI.

2) Hardware Design of PaCC Codec: Below, we present the hardware design of PaCC Codec. Fig. 6 shows the block diagram of our PaCC encoder, which is the hardware realization of PRLE. It consists of an input-end shifting network which shifts n-bit $\mathbf{V}_{\text{diff}}$ from the volatile registers to the RLE encoding module. The n-bit output is the shifted value

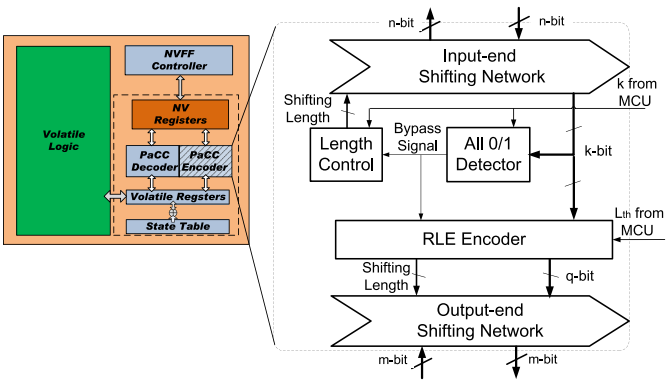


Fig. 6. Block diagram of PaCC encoder and its location in the PaCC architecture.

for updating the volatile registers. Similarly, the output-end shifting network shifts the m-bit compression results to the NV registers. Besides the shifting networks, the all 0/1 detector block helps to execute k-bit parallel observation and generate a bypass signal to the RLE encoder and length controller. The length controller provides the shifting length to the input-end shifting network according to the bypass signal as well as OWW k. The RLE encoder compresses the k-bit input serially when the bypass signal is disabled, otherwise bypasses the k-bit input. The format of the compression result is based on the given L_{th} (see Fig. 5). The OWW k and threshold L_{th} are given by the microcontroller unit (MCU) based on actual applications. Once the RLE encoder accomplishes the compression, it sends the q-bit compressed segment (see Fig. 5) to the output-end shifting network.

The PaCC decoder is similar to the encoder. Since encoding and decoding are opposite operations, the decoder can reuse the two shifting networks. The input-end and output-end parts are exchanged and the data flows in the opposite direction. The only difference in the PaCC decoder is that it contains an RLE decoding module instead of the RLE encoding module. Thus we omit detailed discussion on the decoder part.

In the PaCC codec, the two shifting networks present a big design challenge because the shifting length changes. Though a barrel shifter can handle this task, its area is too large. For example, in the case study to be discussed in Section V, the 8051 processor has $\mathbf{V}_{\text{diff}}$ with 1607 bits. With the shifting length ranging from 1 to 16, the number of multiplexors in the input-end shifting network is $1607 * \log_2(16) = 6428$, which takes up an area larger than the area saved from compression.

To address the area challenge, we propose a new area-efficient shifting structure. Observing that the RLE encoder only deals with the first k bits from the shifting network, we treat the first k bits and the rest of the bits differently. Specifically, we design a two-stage hierarchical shifting network. The structure of our shifting network is shown in Fig. 7. (We use the input-end shifting network as an example since the output-end one has the same structure.) The first stage is a coarse-grained shifting network with a fixed shifting length N. That is, the input in simply shifted by N (i.e, the sth bit becomes the $(N + s)$th bit). This shifter is extremely area-efficient, and no multiplexor is needed (see the upper right for

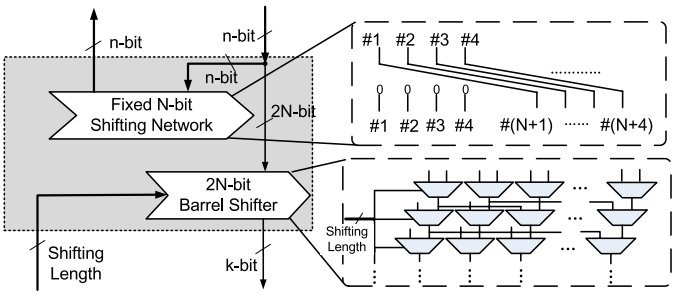


Fig. 7. Structure of the two-stage shifting network.

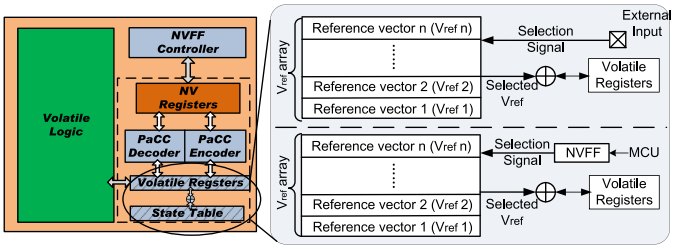


Fig. 8. State table structure and the two proposed reference vector selection methods.

the schematic of this shifter). The second stage is a $2N$-bit barrel shifter which takes the first $2N$ bits of the n-bit input data as its input. Thus, the data to be compressed is shifted by the second stage, and the first stage is used to update the remaining part of the volatile registers. We use a width of $2N$ instead of N to avoid compression performance loss caused by shifting length mismatch. Obviously, the first stage only consumes some interconnects but no multiplexor, and the second stage consumes $2N \times \log_2(2N)$ multiplexors. If N is small, the area can be dramatically reduced.

The determination of N is a tradeoff between area and performance. Basically, N should be small to save area but still maintain the compression performance. At the input end, N must satisfy $N \geq k$. Because k is a variable, N can be set to $\max\{k\}$. At the output end, N should satisfy $N \geq q$, so it can be set to $\max\{q\}$. Note that the values $\max\{k\}$ and $\max\{q\}$ are generally much smaller than n (e.g., in our studied NV processor, $n = 1607$, $\max\{k\} = 16$, and $\max\{q\} = 21$), so the barrel shifter area can be greatly reduced.

B. State Table Design

The state table stores and provides $\mathbf{V}_{\text{ref}}$ for generating $\mathbf{V}_{\text{diff}}$. Since different applications may have very different $\mathbf{V}_{\text{ref}}$s, only one $\mathbf{V}_{\text{ref}}$ is not sufficient. The state table thus contains multiple $\mathbf{V}_{\text{ref}}$s as well as a selection mechanism.

The overall design of the state table is shown in Fig. 8, which consists of a reference vector array and a selection unit. Since encoding and decoding use the same $\mathbf{V}_{\text{ref}}$, the choice of $\mathbf{V}_{\text{ref}}$ should be retained when the power is off. To meet this requirement, we propose two methods. One method adopts external inputs such as switches to hold the choice of $\mathbf{V}_{\text{ref}}$. This method simplifies hardware design and control, but the selection is not very flexible. The other method uses additional

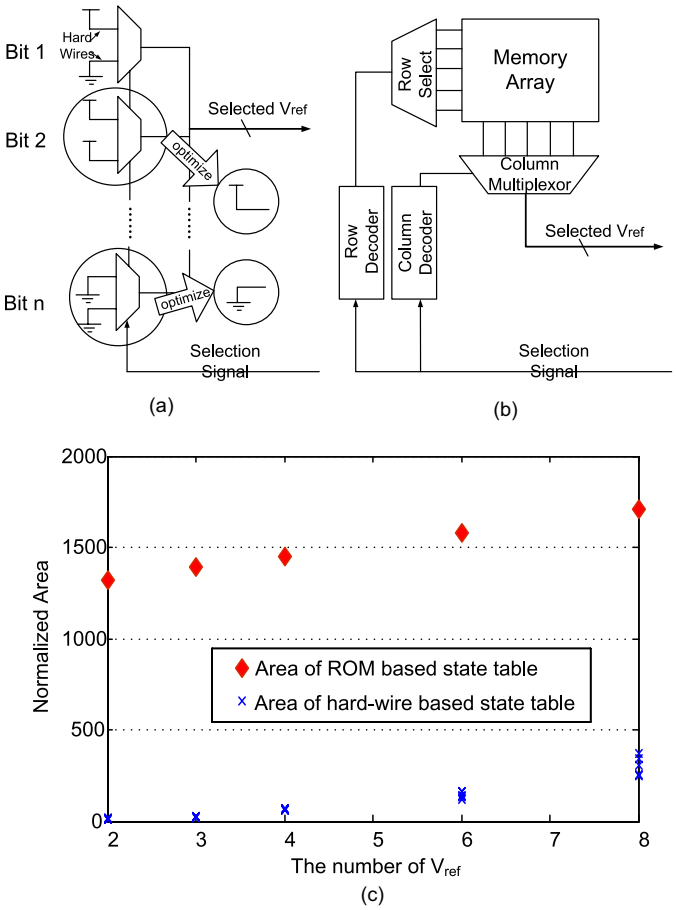


Fig. 9. Two implementations of the reference vector array and their area comparison. The values are normalized to the area of an XOR gate. (a) Hard-wire based reference vector index. (b) ROM based reference vector index. (c) Area comparison of the ROM and hard-wire structures.

NV storage (e.g., 2–3-bit NVFF) to memorize the choice of $\mathbf{V}_{\text{ref}}$ generated dynamically from the MCU according to the actual application. This method requires more complicated software control, but achieves more flexible selection. In real cases, we can use a hybrid selection mechanism combining these two methods such that, if the MCU only runs one application, we can use the external switch method to reduce the control complexity while in other cases we use the dynamic NVFF-based selection.

The reference vector array can be implemented either with a hard-wired structure or ROM because $\mathbf{V}_{\text{ref}}$s are optimized and predefined off line. In the hard-wired structure [shown in Fig. 9(a)], the value of each $\mathbf{V}_{\text{ref}}$ is achieved in the form of hard wires connected to VDD or GND. The selection of $\mathbf{V}_{\text{ref}}$s is made via a set of multiplexors. In this realization, the area overhead is from the multiplexors and the number of $\mathbf{V}_{\text{ref}}$s that can be optimized based on the correlation between $\mathbf{V}_{\text{ref}}$ [as shown in Fig. 9(a), the multiplexor of two 1s or two 0s can be deleted]. This structure is area-efficient when the number of $\mathbf{V}_{\text{ref}}$s is small (only hard wires for one $\mathbf{V}_{\text{ref}}$). However, the hard-wired structure becomes more complex when the number of $\mathbf{V}_{\text{ref}}$ increases.

The ROM-based implementation of the reference vector array [shown in Fig. 9(b)] is more straightforward. It includes

TABLE II
AREA DENOTATION FOR MODULES IN PaCC

Module Name	Denotation
NV registers in conventional NV Processor	$A_{ori,nvr}$
NV registers in PaCC	$A_{pacc,nvr}$
Volatile registers in PaCC	$A_{pacc,vr}$
State table	A_{st}
Bitwise Xor	A_{bxor}
RLE encoder&decoder	$A_{rle,codec}$

a memory array, a row and column decoder, and multiplexors. The ROM-based design is area-wasteful when the number of $\mathbf{V}_{\text{ref}}$s is small, because the decoding circuit and the multiplexors occupy a large portion of the area.

Fig. 9(c) shows quantitative area comparison between hard-wired and ROM designs under different number of $\mathbf{V}_{\text{ref}}$. The areas for the ROM and hard-wired designs are obtained from the ARM-Artisan memory generator and Synopsys Design Compiler, respectively. For a certain number of $\mathbf{V}_{\text{ref}}$, the area of the ROM is a constant value, while the area of the hard-wired structure varies in a small range under different $\mathbf{V}_{\text{ref}}$s. From the figure, we can see that the area of ROM is more than 100× larger than the hard-wired one when the number of $\mathbf{V}_{\text{ref}}$ is 2–4. Even when the number of $\mathbf{V}_{\text{ref}}$ is 8, the area of the hard-wired structure is still 4–5× smaller. Since the number of $\mathbf{V}_{\text{ref}}$ is small in most embedded applications, the hard-wired structure is preferred in our design.

C. Discussion of PaCC Area

In this subsection, we give the area estimation of each module in PaCC and derive the total area reduction that our PaCC architecture can achieve. Generally, the area reduction A_{red} can be expressed as

$$A_{\text{red}} = A_{\text{ori.nvp}} - A_{\text{pacc,nvp}} \tag{2}$$

where $A_{\text{ori,nvp}}$ denotes the area of the NV processor without compression, and $A_{\text{pacc,nvp}}$ denotes the area of the NV processor with PaCC. At the basic module level, A_{red} can be expressed as

$$\begin{aligned} A_{\text{red}} &= A_{\text{ori.nvp}} - A_{\text{pacc,nvp}} \\ &= A_{\text{ori,nvr}} - A_{\text{pacc,nvr}} - A_{\text{pacc,vr}} \\ &\quad - A_{\text{codec}} - A_{st} - A_{b\text{xor}}. \end{aligned} \tag{3}$$

The symbols in (3) are explained in Table II.

As shown in Fig. 3, the reduced part in the PaCC architecture is the number of NV registers. However, the additional parts include the volatile registers, the state table, the bitwise XORs, and the PaCC codec. For the state table estimation, we only consider the area of the V_{ref} array since the remaining parts only contain a few NVFFs and the area is negligible. The area of V_{ref} array is determined by the number of $\mathbf{V}_{\text{ref}}$s. The area of PaCC codec can be obtained by synthesis. The area of the volatile registers and bitwise XORs can be easily calculated, as they are just a series of standard cells. The number of the NV registers in PaCC should be determined under the constraint of compression error rate, which will be discussed in the next section.

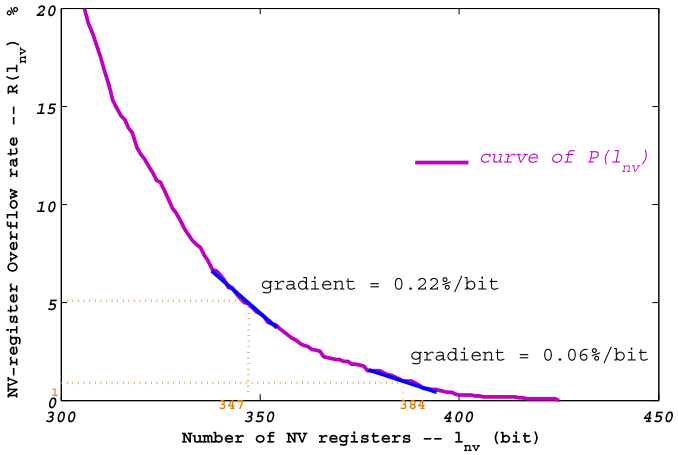


Fig. 10. Overflow rate versus number of NV registers in program Fibonacci.

IV. PaCC TUNING

In the previous section, we discussed the hardware realization of PaCC. In this section, we discuss how to tune the design of PaCC to achieve the best performance. Specifically, the tuning part includes:

1) error tolerance control for handling the NV-register overflow in PaCC;
2) selection of reference vectors for a single application and multiple applications.

A. Overflow Handling and Tolerance Mechanism

In this part, we will firstly explain the cause of compression overflow, and then discuss its tolerance mechanism.

1) Number of NV Registers and Overflow Rate: In PaCC, the NV registers are used to store the system state after compression. Their total number cannot be too large because of the area consideration. However, the limited number may not accommodate all the bits needed by the processor states. When the state vectors do not have good compression ratios, their compression result $\mathbf{V}^c_{\text{diff}}$s can exceed the number of the NV registers. We refer to this scenario as NV-register overflow.

Intuitively, the overflow rate may increase when the number of the NV registers decreases. To quantify their relationship, we introduce a function $R(l_{nv})$ which represents the overflow rate as a function of the number of the NV registers, l_{nv}. To obtain the $R(l_{nv})$, we simulate an example program "Fibonacci" (see program description in Section VI) on an 8051 microprocessor, and sample the register values cycle by cycle as all the possible state vectors, and then calculate the length of $\mathbf{V}_{\text{comp}}$s from all the state vectors. Therefore, we can finally draw the $R(l_{nv})$ curve, shown in Fig. 10, from the length of $\mathbf{V}_{\text{comp}}$s. In Fig. 10, the gradient of the curve indicates the effectiveness of the number of the NV registers in compensating for the overflow rate. When the overflow rate gets lower, the gradient becomes smaller, which means

TABLE III
SYMBOLS USED IN THE REFERENCE VECTOR SELECTION DESCRIPTION

Symbols	Description
$\mathbf{V}_i$	The ith sampled state vector in $\{\mathbf{V}_1,\mathbf{V}_1,...,\mathbf{V}_\delta\}$
$\mathbf{V}_{ref}$	A reference vector for the application.
$\mathbf{V}_{diff}$	The differential vector equals to $\mathbf{V}_i \oplus \mathbf{V}_{ref}$
$\mathbf{V}_{comp}$	The vector to which $\mathbf{V}_{diff}$ is compressed by Algo 1
$\mathbf{V}_{ref,opt}$	The optimal reference vector
$\widetilde{\mathbf{V}}_{ref,opt}$	The sub-optimal reference vector from approximation
$\mathbf{V}_{ref,sub}$	The sub-optimal reference vector from the proposed heuristic
$R(\mathbf{V}_{ref}, l_{nv})$	Overflow rate for a reference vector and # of NV reg.
$L_{nv}(\mathbf{V}_{ref})$	Desired number of NV registers for a reference vector
$\widetilde{L}_{nv}(\mathbf{V}_{ref})$	The number of NV registers from approximated formulation
$L(C(\mathbf{V}))$	The length of vector $\mathbf{V}$ after comparison and compression

more bits are needed to reduce the overflow rate by the same amount. Therefore, it is inefficient to increase the NV registers when $R(l_{nv})$ is very small. As shown in Fig. 10, at point $(347, 5)$, a 1-bit increase can reduce the overflow rate by 0.22% while at point $(384, 1)$, a 1-bit increase can only bring 0.06% overflow reduction which is rather expensive. To prevent unwise increase of the NV registers, we propose to select the designed the number of NV registers, L_{nv}, as follows:

$$L_{nv} = \min\{l_{nv} | P(l_{nv}) \leq \varepsilon\}. \tag{4}$$

That is, L_{nv} captures the transition point where the overflow rate passes ε/bit.

2) Overflow Tolerance Mechanism: As discussed above, given a value of $R(l_{nv})$ determined by (4), there is a small possibility that NV-register overflow may happen during program execution. When this occurs, the system cannot back up the current state to the NV registers. Although the error correction mechanisms, such as ECC, can help recover the overflow error, it requires large area overheads. In this paper, we propose a more efficient tolerance mechanism. We have seen that the hybrid NVFF (shown in Fig. 1) contains a volatile flip-flop and an NV storage. The data is first written in the volatile flip-flop, and then the NV controller backs up the data into the NV storage. When an overflow is detected, the volatile register can send an overflow signal to the NV controller. The controller then suspends its backup operation, and the previous state is kept in the NV storage untouched. On recovery, the processor rolls back to its previous stored state. Although this mechanism leads to some performance loss, it is acceptable when overflows rarely happen.

B. Reference Vector Selection

From the previous discussion, one can see that the CR and the final number of the NV registers strongly depend on the reference vector $\mathbf{V}_{\text{ref}}$ because it decides the distribution of consecutive 0s/1s in $\mathbf{V}_{\text{diff}}$. Since the compression results for different processor states under different $\mathbf{V}_{\text{ref}}$s can be different, finding $\mathbf{V}_{\text{ref}}$s to achieve the best CRs for all states is a nontrivial task. Observing that in a single application program the processor state is relatively stable, we can use one $\mathbf{V}_{\text{ref}}$ for it. Therefore, we discuss the reference vector selection in two stages: 1) optimize the $\mathbf{V}_{\text{ref}}$ in the single application case and 2) determine the $\mathbf{V}_{\text{ref}}$s for multiple applications. In order to make the following description more understandable, we summarize the notation of the vectors and functions in Table III.

1) Single Application Case: When considering the impact of $\mathbf{V}_{\text{ref}}$, the overflow rate $R(l_{nv})$ and the proposed number of NV registers L_{nv} should include the variable $\mathbf{V}_{\text{ref}}$. We extend our earlier notation to represent the overflow rate as $R(\mathbf{V}_{\text{ref}}, l_{nv})$ and the number of NV registers as $L_{nv}(\mathbf{V}_{\text{ref}})$. Before we present the method to find the optimal $\mathbf{V}_{\text{ref}}$, we first define the problem formally. The optimal $\mathbf{V}_{\text{ref}}$, i.e., $\mathbf{V}_{\text{ref,opt}}$, should satisfy the following expression:

$$\begin{aligned}\mathbf{V}_{\text{ref,opt}} &= \arg\min_{\mathbf{V}_{\text{ref}}} L_{nv}(\mathbf{V}_{\text{ref}}) \\ &= \arg\min_{\mathbf{V}_{\text{ref}}}(\min\{l_{nv} | P(\mathbf{V}_{\text{ref}}, l_{nv}) \leq \varepsilon\}).\end{aligned} \tag{5}$$

Unfortunately, because of the complex dependencies of $R(\mathbf{V}_{\text{ref}}, l_{nv})$ on $\mathbf{V}_{\text{ref}}$ and l_{nv} and the huge design space, it is rather difficult to solve (5) directly. In order to determine $\mathbf{V}_{\text{ref,opt}}$, we adopt two simplifications. First, we only sample a limited number of state vectors $\{\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_\delta\}$. Second, we use $\widetilde{L}_{nv}(\mathbf{V}_{\text{ref}})$ to approximate $L_{nv}(\mathbf{V}_{\text{ref}})$, where $\widetilde{L}_{nv}(\mathbf{V}_{\text{ref}})$ indicates the largest length of $\mathbf{V}_i$ after compression. That is

$$\widetilde{L}_{nv}(\mathbf{V}_{\text{ref}}) = \max_{i=1}^{\delta} L(C(\mathbf{V}_i \oplus \mathbf{V}_{\text{ref}})) \tag{6}$$

where operator $\oplus$ represents the bitwise XOR operation. Therefore, (5) can be rewritten as

$$\widetilde{\mathbf{V}}_{\text{ref,opt}} = \arg\min_{\mathbf{V}_{\text{ref}}}(\max_{i=1}^{\delta} L(C(\mathbf{V}_i \oplus \mathbf{V}_{\text{ref}}))). \tag{7}$$

Though (7) is easier to deal with than (5), solving it is still not as straightforward because, to determine a $\mathbf{V}_{\text{ref}}$, $2^{|\mathbf{V}_{\text{ref}}|}$ values need to be checked, which requires tremendous searching time. Moreover, $L(C(\mathbf{V}))$ does not have an analytical expression. Our previous work [22] proposes a majority-voter heuristic which attempts to improve CR by generating most 0s in $\mathbf{V}_{\text{diff}}$. However, it does not differentiate consecutive and nonconsecutive 0s in $\mathbf{V}_{\text{diff}}$ and hence can result in poor CRs for certain $\mathbf{V}_{\text{diff}}$s.

To improve the quality of $\mathbf{V}_{\text{ref}}$, we devise an improved heuristic which gives preferences to consecutive 0s in $\mathbf{V}_{\text{diff}}$. The heuristic relies on a specific directed acyclic graph, $G(\mathcal{V}, \mathcal{E})$, which is obtained based on the bit values in all the state vectors $\{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_\delta\}$. Specifically, each bit of the state is associated with two nodes, one labeled with 0 and the other with 1. Two edges are drawn from each node to the two nodes corresponding to the next bit in the state. For the example vectors in Fig. 11, the state contains 4 bits so there are 8 nodes and 16 edges in $G(\mathcal{V}, \mathcal{E})$. Each edge is assigned a weight which represents the number of state vectors having the corresponding bit transition. For example, among the four vectors in Fig. 11, there is one vector ($\mathbf{V}_2$) whose bit 1 is 0 and bit 2 is 0, and two vectors ($\mathbf{V}_1$ and $\mathbf{V}_3$) whose bit 1 is 0 and bit 2 is 1. Intuitively, if we use the nodes along the longest path from bit 1 to the last bit to construct the reference vector, we would have more $\mathbf{V}_{\text{diff}}$s with consecutive 0s or 1s. This graph-based approach is described in detail in Algorithm 2.

Algorithm 2 Graphic-Based Heuristic for Reference Vector Optimization

Input: $\{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_\delta\}$;
Output: $\mathbf{V}_{ref,sub}$;
Variables: $G(\mathcal{V}, \mathcal{E}), W(\mathcal{E})$;

1 **for** $i \leftarrow 1$ *to* n **do** /* Define nodes */
2 $\quad \mathcal{V}_{i,1} = 0$;
3 $\quad \mathcal{V}_{i,2} = 1$;
4 **for** $j \leftarrow 1$ *to* δ **do** /* Define edge weights */
5 $\quad$ **for** $i \leftarrow 1$ *to* $n-1$ **do**
6 $\quad\quad$ **if** $\boldsymbol{V}_j(i) = 0$ && $\boldsymbol{V}_j(i+1) = 0$ **then**
7 $\quad\quad\quad W(\mathcal{E}_{\mathcal{V}_{i,1},\mathcal{V}_{i+1,1}})$++;
8 $\quad\quad$ **if** $\boldsymbol{V}_j(i) = 0$ && $\boldsymbol{V}_j(i+1) = 1$ **then**
9 $\quad\quad\quad W(\mathcal{E}_{\mathcal{V}_{i,1},\mathcal{V}_{i+1,2}})$++;
10 $\quad\quad$ **if** $\boldsymbol{V}_j(i) = 1$ && $\boldsymbol{V}_j(i+1) = 0$ **then**
11 $\quad\quad\quad W(\mathcal{E}_{\mathcal{V}_{i,2},\mathcal{V}_{i+1,1}})$++;
12 $\quad\quad$ **if** $\boldsymbol{V}_j(i) = 1$ && $\boldsymbol{V}_j(i+1) = 1$ **then**
13 $\quad\quad\quad W(\mathcal{E}_{\mathcal{V}_{i,2},\mathcal{V}_{i+1,2}})$++;
14 $Path = Longest(G)$;
/* Since G is a directed acyclic graph so its longest path can be found in linear time [23] */
15 **for** $i \leftarrow 1$ *to* n **do**
16 $\quad$ **if** $Path(i) == 0$ **then**
17 $\quad\quad \boldsymbol{V}_{ref,sub}(i) = 0$;
18 $\quad$ **else**
19 $\quad\quad \boldsymbol{V}_{ref,sub}(i) = 1$;

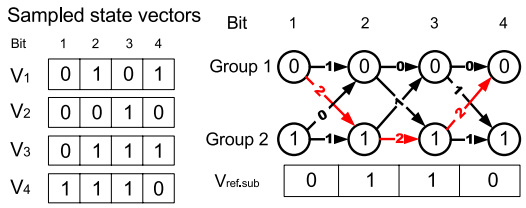


Fig. 11. Illustration of graph-based heuristic.

In Algorithm 2, we first define the node of the graph. For each bit position, we set up two nodes. Node $\mathcal{V}_{i,1}$ denotes the group of vectors whose ith bit is 0, and $\mathcal{V}_{i,2}$ denotes the group of vectors whose ith bit is 1 (Lines 1–3). Then we define the edge weight. The edge weight between $\mathcal{V}_{i,1}$ and $\mathcal{V}_{i+1,1}$, $W(\mathcal{E}_{\mathcal{V}_{i,1},\mathcal{V}_{i+1,1}})$, is set to the number of vectors with the bit value $\mathcal{V}_{i,1}$ and $\mathcal{V}_{i+1,1}$ (Lines 4–13). The longest path from the first bit to the last bit is then computed (Line 14) and the suboptimal reference vector $\mathbf{V}_{\text{ref,sub}}$ is constructed from the longest path (Lines 15–19). For the example in Fig. 11, the resultant reference vector, $\mathbf{V}_{\text{ref,sub}}$, is shown at the lower right.

To help evaluate the quality of the proposed heuristic, we introduce a lower bound on L_{nv}. A tighter low bound is more efficient in assessing the quality of $\mathbf{V}_{\text{ref}}$. To obtain a tight lower bound, we observe that the RLE algorithm

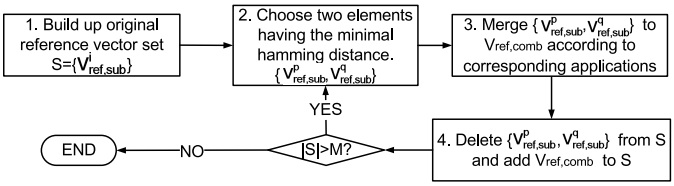


Fig. 12. Reference vector selection flow in the multiple applications case.

Algorithm 3 Reference Vector Merging for Multiple Applications

Input: $\mathbf{V}^p_{ref,sub}, \mathbf{V}^q_{ref,sub}$;
Output: $\mathbf{V}_{ref,comb}$;

1 **for** $i \leftarrow 1$ *to* n **do**
2 $\quad$ **if** $\boldsymbol{V}^p_{ref,sub}(i) == \boldsymbol{V}^q_{ref,sub}(i)$ **then**
3 $\quad\quad \boldsymbol{V}_{ref,comb}(i) = \boldsymbol{V}_{ref,p}(i)$;
4 $\quad$ **else**
5 $\quad\quad \boldsymbol{V}_{ref,comb}(i)$ *equals the majority element of the* i*-th bit of state vectors from application* p *and* q;

can only reduce the number of bits in a long 0/1 sequence while it actually increases the number of bits for other bit patterns. Therefore, a reasonable tight lower bound can be computed as

$$LB(L_{nv}) = n - \sum_{i=1}^{\lambda}(N_i - L_{\text{header}} - \lceil[\log_2(N_i)\rceil) \quad (8)$$

where n denotes the length of the original vector, λ denotes the number of long 0/1 sequences, N_i denotes the length of the ith long sequence, L_{header} denotes the length of the segment head (the part except segment body) in Fig. 5, and $\lceil\log_2(N_i)\rceil$ is the bit width of N_i. A sequence is considered to be long if it contains at least nine consecutive 0s or 1s. We evaluate the tightness of this lower bound and use it to measure the effectiveness of our $\mathbf{V}_{\text{ref}}$ selection heuristic in Section VI.

2) Multiple-Applications Case: Often, a single reference vector cannot achieve high compression ratios for multiple applications, thus multiple reference vectors are preferred. Considering the area overhead, we should not adopt a large number of $\mathbf{V}_{\text{ref}}$s. Given a fixed number of $\mathbf{V}_{\text{ref}}$s as M, if the number of applications is smaller than M, we can simply allocate $\mathbf{V}_{\text{ref,sub}}$ to each application. If he number of applications is larger than M, at least one $\mathbf{V}_{\text{ref}}$ is allocated to two or more applications. In this case, finding the optimal M $\mathbf{V}_{\text{ref}}$s is again a challenging problem.

We introduce a simple solution by merging some of the single-application $\mathbf{V}_{\text{ref,sub}}$s. Specifically, we use the Hamming distance for pairwise $\mathbf{V}_{\text{ref}}$s as the criterion for the merging operation. The approach is depicted in Fig. 12. The key step in the approach is step 3, namely reference vector merging, which is described in Algorithm 3 in detail. Though not optimal, the method is quite effective compared with random choices. Experiments in Section VI confirm this point.

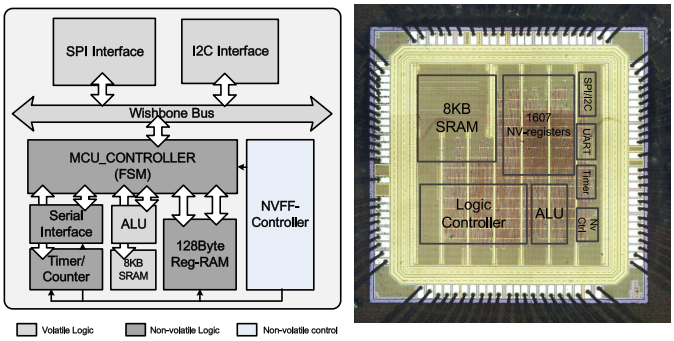


Fig. 13. Micrograph of the fabricated FeFF-based nonvolatile processor.

TABLE IV
NORMALIZED AREA (IN XORS) OF EACH COMPONENT BEFORE AND AFTER DFF REPLACED BY NVFF IN THE STUDIED PROCESSOR

Module Name		Before Replacement (Volatile processor)	After Replacement (NV processor)
MCU	MCU Controller	3734.7	10198.6
	ALU	696.5	696.5
	UART	605.0	605.0
	Timer/Counter	322.1	922.8
	RAM	2585.6	18238.2
SPI/I2C Controller	SPI Controller	300.3	300.3
	I2C Controller	700.8	700.8
NVFF Controller		0	413.5
SRAM		16584.2	16584.2
JTAG		187.2	187.2
I/O Pad		43.5	43.5
Total Chip		25760.0	48890.8

V. CASE STUDY OF NONVOLATILE PROCESSORS

In this section, we present an implementation of an NV processor to help evaluate the PaCC architecture. The processor is a ferroelectric flip-flop (FeFF) based 8051 along with some peripherals such as SPI/I^2C, UART interfaces, etc. We designed and fabricated the NV processor under Rohm's 0.13-μm ferroelectric process [4]. By modeling the FeFF as a standard cell, we used Synopsys synthesis (Design Compiler), place and routing (P&R) tools to realize the whole design. The block diagram and micrograph of the chip is shown in Fig 13.

In order to retain the processor state, all registers and 128-byte internal SRAM are replaced with NVFFs. The total number of NVFFs reaches 1607. To show the area distribution in the processor, we normalize the area of each component to one XOR gate and summarize them in Table IV. In the table, each row corresponds to one component. Columns 3 and 4 present the area before and after NVFF replacement, respectively. As can be seen, the full replacement strategy increases the chip area by nearly 90%. Since the area overhead is very high, this chip is an excellent example to study the merit of the PaCC architecture. Our experiments are based on this NV processor data.

TABLE V
EXPERIMENTAL BENCHMARKS

Application programs	Code Size (Byte)	Memory Occupation (Byte)	Description
Fibonacci	300	24	Fibonacci series generation
Sorting	538	24	Simple sorting of integer series
Sqrt	1133	33	Square root of integers
Rijndael	5545	1378	AES encryption using the Rijndael algorithm
FFT	3495	16	128-point Fast Fourier Transform
Zigbee-MAC	17920	3755	802.15.4 compatible MAC protocol

TABLE VI
NORMALIZED AREA FOR MODULES IN PaCC

Area Denotation	Synthesized Area (Normalized)
$A_{conv,nvr}$	24873.0
$A_{pacc,nvr}$	$15.5*\lvert V_{comp}\rvert$
$A_{pacc,vr}$	4974.6
A_{st}	67.9
A_{bxor}	1607.0
$A_{rle,codec}$	2168.4

[1] The synthesized values are obtained from Synopsys Design Compiler under ROHM's 0.13 CMOS process.
[2] The area values are normalized to the area of an XOR gate.

VI. EVALUATION RESULT

In the evaluation, we implement the PaCC architecture on the 8051 NV processor. We use the behavior simulator Cadence NC-Verilog to sample the system state vectors and evaluate the clock cycles statistics. The area statistics is obtained from Synopsys Design Compiler under Rohm's 0.13-μm ferroelectric-CMOS hybrid process. To simulate the processor behavior in real embedded applications, we use the benchmark program of Fibonacci, sorting and square root from Dalton Project [24], Rijndael and FFT from MiBench [25], and the Zigbee MAC protocol from Z-Stack [26]. The properties of used benchmarks are summarized in Table V.

A. Area Reduction

In order to calculate the area reduced by PaCC as expressed in (3), we have synthesized all the modules included with the Synopsys Design Compiler and listed their area values in Table VI. The numerical values are normalized to the area of one XOR gate. Note that the area of the state table is the typical value of hard-wired structure when the number of $\mathbf{V}_{\text{ref}}$ is 4 (see Fig. 9).

Based on the area data, we can evaluate the area efficiency of PaCC for the programs in Table V. We randomly select 50 state vectors for each program and calculate the suboptimal reference vector $\mathbf{V}_{\text{ref,sub}}$ based on Algorithm 2. According to (6), we get the desired number of NV registers L_{nv}, and the area reduction results in Table VII. In Table VII, each row represents the data for one program and the columns present the data of the optimal threshold L_{th}, compression ratio of PRLE, the number of NV registers, lower bound evaluation on L_{nv}, and area reduction ratio of MCU (only considering the MCU part in Table IV) and the total chip. All the data are obtained under the optimal threshold L_{th} for each program.

TABLE VII
EVALUATION OF AREA EFFICIENCY OF PaCC ARCHITECTURE IN THE SINGLE APPLICATION CASE

Program	Optimal L_{th}	Compression Ratio	# of NV registers	Lower bound on L_{nv}	Area Reduction Ratio	
					MCU only	Total chip
Fibonacci	9	23.7%	381	357	26.2%	17.3%
Sorting	10	26.0%	417	373	24.3%	16.1%
Sqrt	9	27.1%	435	401	23.4%	15.4%
Rijndael	9	20.3%	325	289	29.4%	19.5%
FFT	10	19.2%	308	274	30.2%	20.0%
Zigbee-MAC	10	25.2%	405	381	25.0%	16.5%

TABLE VIII
COMPARISON OF DIFFERENT SHIFTING NETWORK APPROACHES ON AREA REDUCTION

Program	# of MUX's		Shifting Network Area		PACC Codec Area		Area Saving for PaCC codec	Area Saving for total chip
	1-stage	2-stage	1-stage	2-stage	1-stage	2-stage		
Fibonacci	9940	1200	10471.9	1049.7	11600.5	3340.0	71.2%	18.7%
Sorting	10120	1200	10732.0	1231.7	11860.6	3400.1	71.3%	19.0%
Sqrt	10210	1200	10862.1	1261.7	11990.6	3430.1	71.4%	19.2%
Rijndael	9575	1200	9944.5	1171.6	11073.1	3218.2	70.9%	18.1%
FFT	9445	1200	9756.7	1006.4	10885.2	3174.8	70.8%	17.9%
Zigbee-MAC	10060	1200	10645.3	1211.7	11773.9	3380.1	71.3%	18.9%

[1] The area values are normalized to the area of an XOR gate.

The optimal threshold is the one that results in the smallest number of NV registers among all the threshold values in [4, 50]. We can see that, for the programs considered, the optimal L_{th} is always 9 or 10. This is due to the fixed encoding format shown in Fig. 5.

As shown in Table VIII, different programs may lead to different numbers of NV registers (see col. 4), thus, the area savings vary for different programs. By utilizing PaCC, the compression ratio can reach to 19.2% with the number of NV registers reduced from 1607 to 308. Based on this reduction, the area saving ratio for the MCU can only be 23.4%–30.2%, and for the total chip the worst case ratio is still above 15%. Comparing cols. 4 and 5, we find that the lower bound on L_{nv} is about 7%–10% lower than what our heuristic achieved. Since the bound is based on some extreme assumptions, we can conclude that our algorithm is quite effective.

Table VIII shows the comparison results over the two different shifting network approaches in Fig. 7. The comparison data consider both the input-end and output-end shifting networks (see Fig. 6). From the table, we can see for the programs, the two-stage shifting network can reduce the number of MUXs by 8–9×, which leads to 70.8%–71.4% area saving of the PaCC codec and 17.9%–19.0% area saving of the whole chip compared to signal-stage barrel shifter. This part of experiment results confirms that the two-stage shifting network is much more area-efficient than the conventional barrel shifter.

To evaluate the area contribution of the submodules in PaCC, we normalize the areas of the submodules in PaCC as well as their area in a conventional NV processor to one XOR gate in Fig. 14. Though the area distribution is a little different between programs, we can still see some commonalities. In the conventional NV processor, the NV registers contribute to more than 50% of the total chip area, while in the NV processor with PaCC, the percentage is reduced to less than 20%. Furthermore, the PaCC codec occupies 12%–13% of the total chip area, which is tolerable considering the large savings in the NV registers. The area of the volatile part in the NV processor with PaCC is increased because of the extra volatile registers. Nevertheless, they only lead to 20% area increase in the volatile part. By considering all the factors impacting the processor area, the PaCC can significantly reduce the NV register and the total chip area.

TABLE IX
EVALUATION OF RUN TIME OF PaCC CODEC

Program	Optimal k	Clock Cycles				Process Time on Average (μs)		Performance Overhead (times)	
		Encode		Decode					
		Mean	Std	Mean	Std	Encode	Decode	Backup	Restore
Fibonacci	19	243.2	21.2	97.8	3.3	24.3	9.8	4.5	4.3
Sorting	16	253.1	25.7	98.1	3.3	25.3	9.8	4.6	4.3
Sqrt	17	301.8	34.0	101.0	2.9	30.2	10.1	5.3	4.4
Rijndael	16	211.7	23.4	95.3	3.3	21.1	9.5	4.0	4.2
FFT	20	190.9	28.9	94.5	3.9	19.1	9.5	3.7	4.2
Zigbee-MAC	20	279.5	42.5	98.5	2.2	27.0	9.9	4.9	4.3

[1] Assuming the data encoding and decoding procedures runs at 10MHz clock frequency, the clock cycle statistics is obtained from a circuit simulator.
[2] "Mean" means the average value, "Std" means the standard deviation.
[3] Performance overhead is the time prolonging comparing with conventional NV processor.

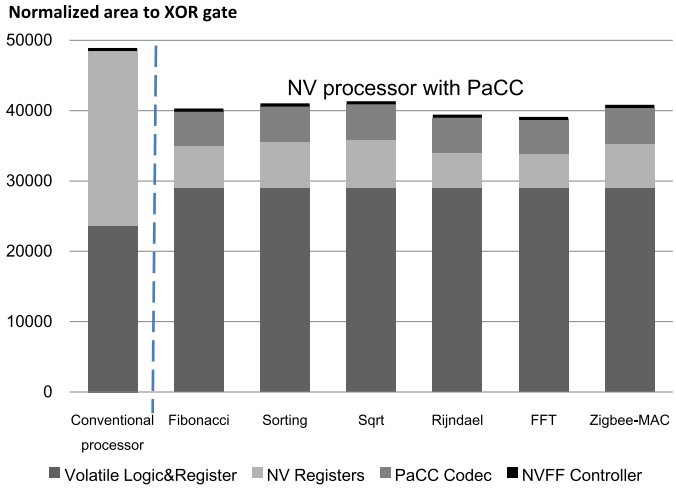


Fig. 14. Area of submodules in the conventional NV processor and the NV processor with PaCC for different programs. The area is normalized with respect to an XOR gate.

B. Codec Performance

The runtime of encoding and decoding is an important metric for the PaCC codec. The encoding performance depends on the chosen OWW, k, so we first decide the appropriate k by some experiments. Fig. 15 shows the clock cycles taken by the encoding process (obtained from RTL simulation by Mentor's Modelsim) versus k for different programs. For each program, we use the average encoding runtime for its state vectors.

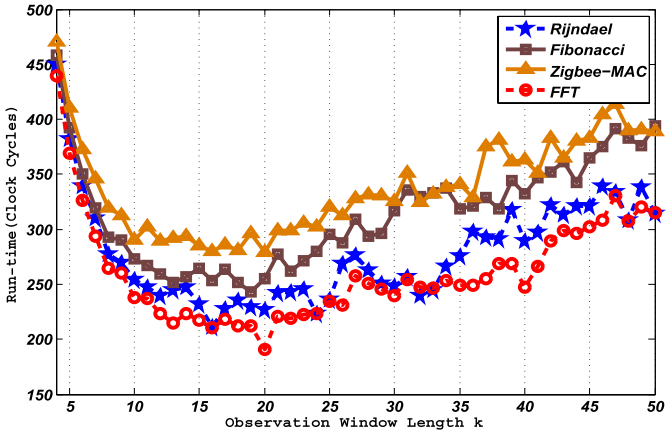


Fig. 15. Codec runtime versus observation window width for different programs.

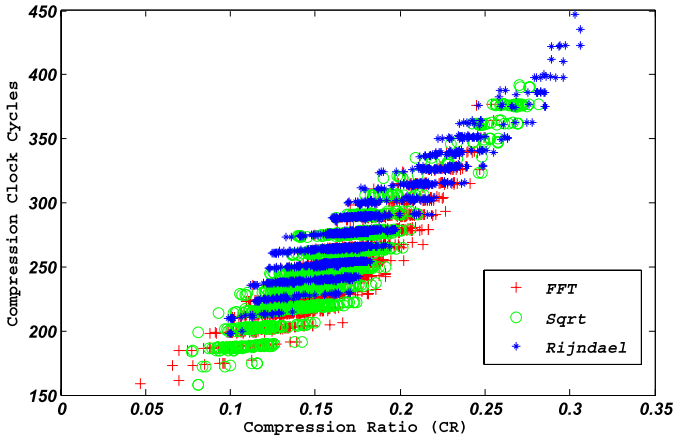


Fig. 16. Relationship of compression ratio and compression performance.

Intuitively, a smaller k may not achieve significant reduction in clock cycles, while a larger k reduces the opportunities of encountering consecutive zeros. As a result, we can get an optimal k that leads to the smallest number of encoding clock cycles. Fig. 15 shows that, though the optimal k may vary for different programs, it usually locates in a fixed range of 16–20.

Given the optimal k chosen for each program, Table IX shows the clock cycles of encoding and decoding for different programs. We can see that the encoding process needs extra 200–300 cycles to compress one vector, while the decoding one costs 90–100 cycles. Therefore, the time of storing data takes less than 30 μs and recalling takes less than 10 μs at the 10-MHz clock frequency. The last two columns show the overall performance overhead of PaCC compared to a traditional NV processor. The values indicate a relatively significant increase in the backup and restore time. The backup and restore time of the conventional NV processor are 3 and 7 μs, respectively, and they are increased by 3–5$\times$ because of the PaCC codec process. However, the total backup and restore time is still less than 50 μs, which still maintains the instant on/off features of the NV processors.

Furthermore, we observe that the CR and performance of PaCC are somewhat related. Intuitively, a better CR implies more zeros in $\mathbf{V}^c_{\text{diff}}$ and more opportunities of parallel operation. Therefore, the CR and compression speed have a certain degree of positive correlation. Fig. 16 shows the distribution of CR and compression speed for sets of input vectors in different applications. The data indicate better compression performance by means of achieving better CR and this correlation is tenable despite variations in the applications.

C. Overflow Rate

In a single application, we determine the number of NV registers and appropriate reference vector based on (4) and (7) to depress the overflow effectively. However, overflow may still exist in some particular situations. To analyze the overflow distribution, we sample the system state at every clock cycle and calculate the probability distribution of $|\mathbf{V}^c_{\text{diff}}|$. Fig. 17 shows the results for the program Rijndael. (We have observed

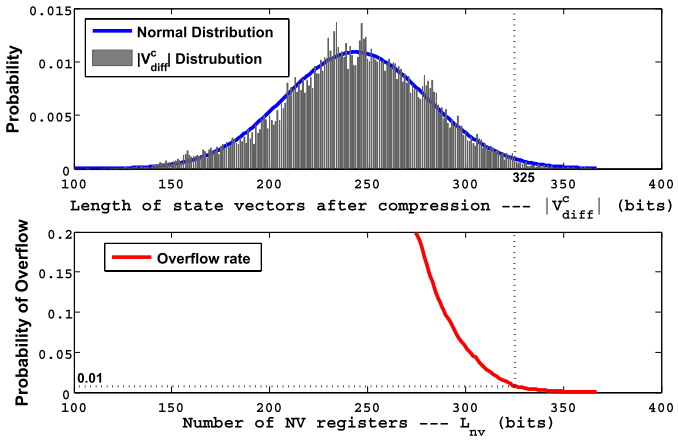


Fig. 17. Possibility distribution and compression overflow ratio for different lengths of $\mathbf{V}^c_{\text{diff}}$.

similar phenomena in other programs.) The probability density approximates a Gaussian distribution, so we can describe it with mean and variance values. The overflow rate function ($R(l_nv)$) can be obtained by integrating the probability density function. In this program, we adopt 325-bit NV registers to make the $R(l_{nv})$ below 1%.

Table X shows for different programs the mean and standard deviation of $|\mathbf{V}^c_{\text{diff}}|$ as well as the desired number of NV registers at certain overflow rate. The result reveals that certain compression variations exist between different system states in a single application. However, the standard deviation is always below 30 bits. Moreover, if the number of NV registers is 1 standard deviation above the mean value, the overflow rate can be reduced to nearly 10%. Comparing with Table VIII, one can see that the L_{th} calculated from 50 sampled system states can reduce the overflow rate to nearly 1%.

D. Reference Vector Selection

Now we evaluate the effectiveness of our heuristics for reference vector selection. Thee evaluation is done for both the simple application case and the multiple application case.

1) Single Application Case: We have compared our proposed heuristic (Algorithm 2) with four other reference vector

TABLE X
EVALUATION OF $|V^c_{\text{diff}}|$ DISTRIBUTIONS AND THE NUMBER OF NV REGISTERS UNDER DIFFERENT OVERFLOW RATE

Program	$\|V^c_{diff}\|$ Distribution		# of NV registers	
	Mean (Bit)	Std (Bit)	10% Overflow Rate	1% Overflow Rate
Fibonacci	316.7	23.2	346	384
Sorting	369.3	15.9	380	416
Sqrt	375.4	22.8	403	439
Rijndael	247.2	29.0	282	325
FFT	245.3	19.8	268	307
Zigbee-MAC	358.3	20.5	381	406

[1] "Mean" means the average value, "Std" means the standard deviation.

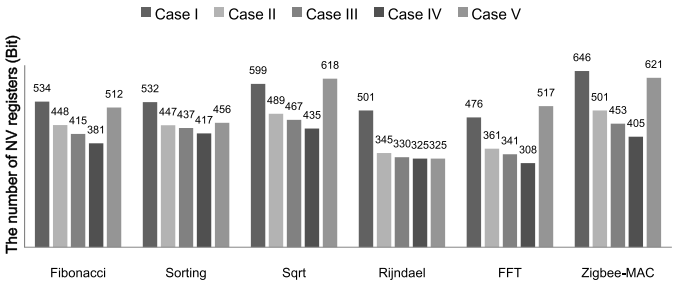


Fig. 18. Comparison of the number of NV registers for different reference vector selection methods in the single application case.

selection methods in term of the number of NV registers after compression. For ease of discussion, we categorize the five $\mathbf{V}_{\text{ref}}$ selection methods as follows.

Case I: all-zero vector.

Case II: randomly chosen state vector $\mathbf{V}_i$.

Case III: $\mathbf{V}_{\text{ref,sub}}$ from method in [22] for each program.

Case IV: $\mathbf{V}_{\text{ref,sub}}$ from Algorithm 2 for each program.

Case V: $\mathbf{V}_{\text{ref,sub}}$ based on a single program Rijndael.

Fig. 18 shows the results for the different programs. Comparisons between Cases I, II, and IV indicates that the reference vector selected by our heuristic (Case IV) can reduce the number of NV registers by as much as 59% while in the worst case it can result in 28% reduction. Comparisons between Cases III and IV reveals that our new heuristic can further reduce the number of NV registers by 2%–12% beyond that obtained by the method from [22]. Case V uses Rijndael as an example to illustrate that a $\mathbf{V}_{\text{ref,sub}}$ proper for one application may not be suitable for other applications. The data indicate that a multiple-application adapted state table is necessary. Furthermore, Case V for program sorting is not very bad, which means that a universal reference vector can be appropriate for a set of applications. The above observation is significant for conducting reference vector selection in the multiple-application case.

2) Multiple-Application Case: In this part, we evaluate the determination of reference vectors $\{\mathbf{V}_{\text{ref}}\}$s in the multiple-application case. We consider three different reference vector choices and their impact on the number of NV registers.

Case I: Use the reference vectors obtained by the technique in Fig. 12.

Case II: Randomly choose M $\mathbf{V}_{\text{ref,sub}}$s and allocate them to the applications randomly.

TABLE XI
EVALUATION OF NUMBER OF NV REGISTERS UNDER DIFFERENT REFERENCE VECTOR SELECTION METHODS IN THE MULTIAPPLICATION CASE

Case	# of NV registers for different number of $\mathbf{V}_{ref}$'s			
	$M=2$	$M=3$	$M=4$	$M=5$
Proposed method	795	616	612	525
Randomly combine two programs	920	906	918	928
Without V_{ref} merge	838	637	627	570

Case III: Use a $\mathbf{V}_{\text{ref,sub}}$ from one application without merging two $\mathbf{V}_{\text{ref,sub}}$s when considering two applications (see Step 3 in Fig. 12).

We only consider the situation that the number of reference vectors is less than that of applications. Table XI shows the experiment result. From every row of the table, we can see that the number of NV registers decreases as the number of $\mathbf{V}_{\text{ref}}$s increases. This is intuitive since more $\mathbf{V}_{\text{ref}}$s mean more opportunities to choose a proper $\mathbf{V}_{\text{ref}}$ for each application. Comparisons between Case I and Case II reveals that our method can reduce the number of NV registers by 43% compared to random selection and allocation. Comparing Case I and Case III reveals that the merging operation can reduce the number of NV registers by 3%–8%. Furthermore, the data in Tables XI and VII show that it is difficult to achieve the same area efficiency for the multiple-application case as for the single application case. However, when the number of $\mathbf{V}_{\text{ref}}$s is not too small, the increase in the number of NV registers is quite acceptable.

E. Discussion of Backup Energy

Similar to the area reduction, the reduction of NVFFs can also help to reduce the backup energy cost. In this subsection, we simply discuss the backup energy reduction in the 8051 NV processor when implementing PaCC.

The total backup energy is mainly comes from three parts: the NVFFs, the NV controller and PaCC encoding. From the real test of the 8051 NV processor, we find the NVFFs consume 21.48 nJ, which is 93% of total backup energy, and the NV controller consumes 1.62 nJ (7%). In PaCC, we assume the energy consumption of the NV controller is invariant, and the energy consumption of NVFFs is proportional to their number. We obtain the energy cost of the PaCC codec from power analysis, which is 3.54 nJ. We assume the number of NVFFs is reduced by θ. Therefore, we can calculate the total backup energy reduction as $(21.48 \times \theta - 3.54)$ nJ. From Table VII, according to the CR, we can easily find θ is at least 72.9%. Therefore, the backup energy is reduced by at least 12.12 nJ, or 52.5% in percentage terms.

VII. CONCLUSION

NV processors, based on emerging technologies, open up new domains for power savings and attractive applications. However, the traditional approach of full register replacement causes nontrivial area overheads and increases the chip cost

significantly. This paper proposed a PaCC architecture to reduce the number of NVFFs. A novel PaCC codec and a reconfigurable state table were introduced, and corresponding heuristics were developed to optimize reference vector selection. Experimental results showed that we could achieve over 70% NVFF reductions and up to 30% overall processor area saving. Meanwhile, the compare and compress process only incurred tens of microseconds time overhead.

Although we have evaluated the PaCC architecture on a ferroelectric-based processor, the proposed approach is applicable to designs based on other NV materials which have nontrivial area overhead due to the use of on-chip NV storage elements. Our future work will focus on the integration of the PaCC architecture into an NV processor for actual chip fabrication and applying it in real embedded systems. Moreover, we plan to extend the PaCC to design NV multicore processors and other complicated systems.

REFERENCES

[1] W. M. Jones, J. T. Daly, and N. DeBardeleben, "Application monitoring and checkpointing in HPC: Looking towards exascale systems," in *Proc. 50th Annu. Southeast Regional Conf.*, 2012, pp. 262–267.

[2] M. Zwerg, A. Baumann, R. Kuhn, M. Arnold, R. Nerlich, M. Herzog, R. Ledwa, C. Sichert, V. Rzehak, P. Thanigai, and B. Eversmann, "An 82 μA/MHz microcontroller with embedded FeRAM for energy-harvesting applications," in *Proc. ISSCC*, Feb. 2011, pp. 334–336.

[3] W. Yu, S. Rajwade, S. Wang, B. Lian, G. Suh, and E. Kan, "A non-volatile microcontroller with integrated floating-gate transistors," in *Proc. 5th Workshop Dependable Secure Nanocomput.*, 2011, pp. 1–4.

[4] Y. Wang, Y. Liu, S. Li, D. Zhang, B. Zhao, B. Sai, M.-F. Chiang, Y. Yan, and H. Yang, "A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops," in *Proc. ESSCIRC*, Sep. 2012, pp. 149–152.

[5] M. Qazi, A. Amerasekera, and A. P. Chandrakasan, "A 3.4 pJ FeRAM-enabled D flip-flop in 0.13 μm CMOS for nonvolatile processing in digital systems," in *Proc. ISSCC*, Feb. 2013, pp. 192–193.

[6] (2007). Rohm Co., Ltd. *Rohm Demonstrates Non-volatile CPU*, Kyoto, Japan [Online]. Available: http://techon.nikkeibp.co.jp/english/NEWS_EN/20071004/140206/

[7] Y. Gang, W. Zhao, J.-O. Klein, C. Chappert, and P. Mazoyer, "A high-reliability, low-power magnetic full adder," *IEEE Trans. Magn.*, vol. 47, no. 11, pp. 4611–4616, Nov. 2011.

[8] *Datasheet of MSP430F522X Mixed Signal Microprocessors*, Texas Instrument, Dallas, TX, USA, 2009.

[9] *Datasheet of AT91SAM9G20-AT91 ARM Thumb Microcontrollers*, Atmel, San Jose, CA, USA, 2012.

[10] M. Poremba and Y. Xie, "NVMain: An architectural-level main memory simulator for emerging non-volatile memories," in *Proc. ISVLSI*, 2012, pp. 392–397.

[11] (2011). ITRS. *Roadmap for Nonvolatile Memory*, Taiwan [Online]. Available: http://www.itrs.net/

[12] W. Zhao, E. Belhaire, V. Javerliac, C. Chappert, and B. Dieny, "A non-volatile flip-flop in magnetic FPGA chip," in *Proc. DTIS*, Sep. 2006, pp. 323–326.

[13] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai, "Nonvolatile magnetic flip-flop for standby-power-free SoCs," *IEEE J. Solid-State Circuits*, vol. 44, no. 8, pp. 2244–2250, Aug. 2009.

[14] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing," in *Proc. 37th Annu. ISCA*, 2010, pp. 371–382.

[15] Nikkei Electronics Asia. (2008). *Rohm Develops Non-Volatile Register; Slashes Dissipation*, Hong Kong [Online]. Available: http://techon.nikkeibp.co.jp/article/HONSHI/20080729/155646/

[16] J. Wang, Y. Liu, H. Yang, and H. Wang, "A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications," in *Proc. ICGCS 2010*, pp. 646–650.

[17] R. Beach, T. Min, and C. Horng, "A statistical study of magnetic tunnel junctions for high-density spin torque transfer-MRAM (STT-MRAM)," in *Proc. IEEE IEDM*, Dec. 2008, pp. 1–4.

[18] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[19] R. Williams, "An extremely fast Ziv-Lempel data compression algorithm," in *Proc. DCC*, Apr. 1991, pp. 362–371.

[20] J. Trein, A. Schwarzbacher, B. Hoppe, and K. Noff, "A hardware implementation of a run length encoding compression algorithm with parallel inputs," in *Proc. ISSC*, Jun. 2008, pp. 337–342.

[21] G. Beenker and K. Immink, "A generalized method for encoding and decoding run-length-limited binary sequences (Corresp.)," *IEEE Trans. Inf. Theory*, vol. 29, no. 5, pp. 751–754, Sep. 1983.

[22] Y. Wang, Y. Liu, Y. Liu, D. Zhang, S. Li, B. Sai, M.-F. Chiang, and H. Yang, "A compression-based area-efficient recovery architecture for nonvolatile processors," in *Proc. DATE*, Mar. 2012, pp. 1519–1524.

[23] R. Uehara and Y. Uno, "Efficient algorithms for the longest path problem," in *Algorithms and Computation*. New York, NY, USA: Springer-Verlag, 2005, pp. 871–883.

[24] (2006, Aug.). *Benchmark Applications for Synthesizeable VHDL Model* [Online]. Available: http://www.cs.ucr.edu/~dalton

[25] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. WWC*, Dec. 2001, pp. 3–14.

[26] Texas Instrument. (2009). *Z-Stack—ZigBee Protocol Stack*, Dallas, TX, USA [Online]. Available: http://www.ti.com/tool/z-stack

Yiqun Wang (S'12) received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2009, where he is currently pursuing the Ph.D. degree with the Electronic Engineering Department.

His current research interests include low power VLSI designs and nonvolatile processors and memories. He is now working in the project of nonvolatile processor design and wireless sensor network SOC design.

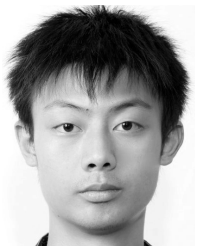
Yongpan Liu (M'07) received the B.S., M.S., and Ph.D. degrees from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 1999, 2002, and 2007, respectively.

He was a Research Fellow with Tsinghua University from 2002 to 2004. He is currently an Associate Professor with the Department of Electronic Engineering, Tsinghua University. His research is supported by NSFC, 863, 973 Program and Industry Companies such as Intel, Rohm, and IBM. He has published over 50 peer-reviewed conference and journal papers and led over six SoC design projects for sensor applications. His current research interests include low power VLSI design, emerging device based circuits, and systems and design automation.

Dr. Liu received the ISLPED in 2012, ISLPED in 2013 Design Contest Award, and several best paper candidates. He has been invited to serve on several conference technical program committees (ASP-DAC, ISLPED, ICCD, A-SSCC).

Shuangchen Li received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2011, where he is currently pursuing the master's degree with the Electronic Engineering Department.

His current research interests include high-level synthesis and low power VLSI designs.



Xiao Sheng received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2012, where he is currently pursuing the master's degree with the Electronic Engineering Department.

He is now working in the project of an application of nonvolatile wireless sensor network in bridge health monitor. His current research interests include low power VLSI designs and nonvolatile processors and memories.

Daming Zhang received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2010, where he is currently pursuing the Ph.D. degree with the Electronic Engineering Department.

He is now working in the project of a self-powered WSN sensor platform design. His current research interests include high-level synthesis and self-powered SoC designs.

Mei-Fang Chiang received the B.S. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2004, and the M.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 2006.

She joined ROHM Co., Ltd., Kyoto, Japan, in 2009, and is currently working on digital circuit design.

Baiko Sai received the B.E. degree from the Shanghai University of Science and Technology, Shanghai, China, in 1985, and the M.E. degree from Yokohama National University, Yokohama, Japan, in 1990.

He was with the Space Science and Technology Institute of China from 1985 to 1987. He was with Pioneer Electronic Corporation, Tokyo, Japan, from 1990 to 1996. He was with Philips Japan Ltd., Tokyo, from 1996 to 2001, and he is currently with Rohm Co., Ltd., Kyoto, Japan. He is a Guest Professor with the Department of Computer Science and Electronics, Kyushu Institute of Technology, Fukuoka, Japan. Since 2013, he has been with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan, as an Associate Professor. His current research interests include wireless communication, digital signal processing, digital broadcasting system, high speed interface, and information security technologies.

Xiaobo Sharon Hu (S'85–M'89–SM'02) received the B.S. degree from Tianjin University, Tianjin, China, the M.S. degree from the Polytechnic Institute of New York, Brooklyn, NY, USA, and the Ph.D. degree from Purdue University, West Lafayette, IN, USA.

She is a Professor with the Department of Computer Science and Engineering, University of Notre Dame, South Bend, IN, USA. She has published more than 200 papers. Her current research interests include real-time embedded systems, low-power system design, and VLSI and nano-scaling computing.

Dr. Hu is currently an Associate Editor for the *ACM Transactions on Embedded Computing*. She served as an Associate Editor for the IEEE TRANSACTIONS ON VLSI and *ACM Transactions on Design Automation of Electronic Systems* and special-issue Guest Editor for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS. She received the NSF CAREER Award in 1997, the Best Paper Award from Design Automation Conference in 2001, and the IEEE Symposium on Nanoscale Architectures in 2009.

Huazhong Yang (SM'13) was born in Sichuan, China, on August 18, 1967. He received the B.S. degree in microelectronics and the M.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively.

He joined the Department of Electronic Engineering, Tsinghua University, Beijing, in 1993, where he has been a Full Professor since 1998. He is a Specially-Appointed Professor of the Cheung Kong Scholars Program. He has authored or co-authored over 300 technical papers and 70 granted patents. His current research interests include wireless sensor networks, data converters, parallel circuit simulation algorithms, nonvaltile processors, and energy-harvesting circuits.