A Compression-based Area-efficient Recovery Architecture for Nonvolatile Processors

Yiqun Wang * Yongpan Liu* Yumeng Liu* Daming Zhang * Shuangchen Li*
Baiko Sai† Mei-Fang Chiang† Huazhong Yang*

{ypliu,yanghz}@tsinghua.edu.cn {baiko.sai,meifang.chiang}@dsn.rohm.co.jp
* TNList, EE Dept.,Tsinghua University † LSI Development Headquarters, Rohm Co., Ltd.
Beijing, 100084, China Yokohama, 222-8575, Japan

***Abstract*—Nonvolatile processor has become an emerging topic in recent years due to its zero standby power, resilience to power failures and instant on feature. This paper first demonstrated a fabricated nonvolatile 8051-compatible processor design, which indicates the ferroelectric nonvolatile version leads to over 90% area overhead compared with the volatile design. Therefore, we proposed a compare and compress recovery architecture, consisting of a parallel run-length codec (PRLC) and a state table logic, to reduce the area of nonvolatile registers. Experimental results demonstrate that it can reduce the number of nonvolatile registers by 4 times with less than 1% overflow possibility, which leads to 43% overall processor area savings. Furthermore, we implemented the novel PRLC and defined the method to optimize the optimal parallel degree to accelerate the compressions. Finally, we proposed a reconfigurable state table architecture, which supports the reference vector selecting for different applications. With our heuristic vector selecting algorithm, the optimal vector can provide over 42% better register number reduction than other vector selecting approaches. Our method is also applicable to designs with other nonvolatile materials based registers.**

I. Introduction

With the emerging memory technologies, nonvolatile processors [1]–[4] are receiving more and more attentions. Compared with the volatile ones, the nonvolatile processors are manufactured with nonvolatile registers and have the following advantages: I) zero-standby power: the processor can retain its state when not powered, while the traditional ones suffer from the increasing leakage power to keep data; II) instant on and off: the processor can resume its work within several cycles from the stalled point, while the traditional one needs millions of initializing cycles; III) high resilience to power failures: the processor can work reliably under the environments with frequency power interrupts, such as energy harvesting and wireless powered applications [2]; IV) fine-grained power management supported [1]: the processor can be shut down whenever possible due to the ultra-low energy and fast recovering characteristics. Therefore, research on nonvolatile processors are interesting.

In order to implement nonvolatile processor, plenty of work has been done on the nonvolatile memory cells [5]. In the realm of random access memory (RAM), Flash is a mature high-density solution. However, it suffers from low endurance and slow writing speed as a register. Phase Change Random Access Memory (PRAM) has the highest potential in density and can be used in a hybrid cache with static RAMs [6], however a longevity around 10^9 cycles also hinders its further application as registers for processor [5]. By contrast nearly unlimited operation cycle and ultra short access time render both Ferroelectric RAM and Magnetic RAM emerge as the most promising candidates. Sakimura et.al. [7] designed a Magnetic Flip-flop for systems-on-chip with measured results. Zhao et.al. [8] applied (Magnetic Tunnel Junction) MTJ-based flip-flops in FPGAs. Rohm [9] developed a nonvolatile register by adding a ferroelectric capacitor to a standard register. However, none of them considers the impact of deploying nonvolatile registers in real processors.

Recently, several nonvolatile digital designs are reported using nonvolatile registers. In [10], people observed a over 40% larger chip area in a low-pass digital filter by replacing traditional registers with Magnetic ones. According to [11], the ferroelectric capacitor based register is over 4 times larger than a normal one and greatly increase the area of a nonvolatile processor. In [2], a nonvolatile controller is evaluated based on the floating-gate transistors. The results indicate over 20% chip area and 40% memory area overheads. Guo et.al [12] had proposed a STT-MRAM based processor with promising area and energy, however the results are not supported by real chips. Generally, the replacement of nonvolatile registers increases the chip area and costs significantly. It is necessary to investigate techniques to realize area-efficient nonvolatile processors.

Surprisingly, previous nonvolatile designs assume a conservative policy: replacing all volatile registers with nonvolatile ones is necessary to store the system state. This assumption would lead to a significant area increase. However, this paper demonstrates that it is possible to store the system state with much fewer nonvolatile registers with the proposed compression-based recovery architecture. Our contributions are listed as the followings:

1) We validated the necessity of area-aware designs by a real nonvolatile processor in Section III and proposed a compression-based state recovery architecture to reduce the number of the nonvolatile registers, and thus the area of the processor. The architecture adopts a compare and compress strategy to improve the compressing ratio. It consists of a compressing codec and a state table logic.
2) In order to shorten the compressing time, we design a parallel run-length codec (PRLC) in Section IV. It can achieve over 5 times speedup in average over the conventional serial RLC.
3) We also provide a reconfigurable architecture to construct the state table, which can be tuned to fit different applications. To maximize the number of zeros after the comparison, we formulate the optimal vector selection problem and develop a heuristic algorithm to solve it.
4) We evaluate the proposed technique in a nonvolatile 8051 processor. Several real programs are used to verify the efficiency of the recovery architecture. The area reduction is up to 43%, while the overall codec time is less than $50\mu s$ under a 10 MHz clock frequency in Section V.

II. Motivation

This section first describes the area challenges from the replacement of nonvolatile registers. After that, an interesting observation is

This work was supported in part by the NSFC under grant 60976032, National Science and Technology Major Project under contract 2010ZX03006-003-01, International Cooperation from ROHM Inc. and High-Tech Research and Development (863) Program under contract 2009AA01Z130.
DATE'12, March 12-16, 2012, Dresden, Germany.

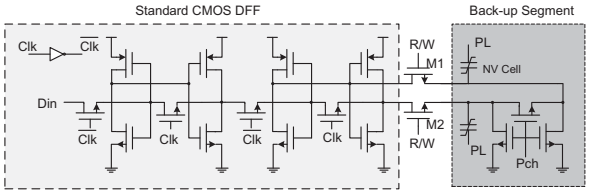

Fig. 1. The Schematic of nonvolatile flip-flop

given out, which motivates the area-efficient recovery architecture.

A. Hybrid Nonvolatile Flip-flop

Fig. 1 shows the schematic of a hybrid nonvolatile flip-flop which consists of a standard master-slave DFF and a backup segment containing nonvolatile components. In the normal mode, the switches M1 and M2 are open and the backup segment is isolated. The flip-flop works as a standard CMOS DFF. The back-up segment is activated only when the DFF state needs to be stored. In those cases, M1 and M2 are short and clock is gated, thus signals PL and Pch control the data storage/recall operations. This nonvolatile flip-flop structure induces no deterioration in normal performance but prolongs the longevity of nonvolatile components.

B. Area Challenges of Nonvolatile Register

Generally, a nonvolatile register should contain new components, such as ferroelectric capacitors, magnetic tunnel junctions or floating-gate transistors. It makes the area of a nonvolatile register much larger than the volatile CMOS register. We define that a nonvolatile register is α times larger than the original one. Assuming the register and memory blocks occupy $\beta(0 < \beta < 1)$ of total chip area, the chip area becomes $S_{ovh} = \beta \times (\alpha - 1)$ larger after replacing all memory units with nonvolatile registers. In the realized 8051 processor, β equals to 20% and α is near to 5, thus $S_{ovh} = 80\%$. Obviously, this area overhead S_{ovh} is unacceptable and we need a method to reduce the area of nonvolatile registers.

C. Compare and Compress Strategy

We define the system state as the state vector $\mathbf{V} = (v_1, v_2, ..., v_n), v_i = 0/1$, where each bit represents the current value of a flip-flop in a processor. In the 8051 processor, n equals to 1607. As replacing all memory units with nonvolatile register would lead to prohibitive area overheads, is it possible to memorize the system state with fewer nonvolatile registers? In the experiments, we observe an interesting phenomenon on the system state of the nonvolatile processor. Over 80% registers are not altered from the reset state $\mathbf{V}_{reset}$ in the real applications. Thus, only a small portion of registers are different from its reset state. Suppose we need to store the system state at the ith breakpoint $\mathbf{V}_i$, we can just store the differential vector $\mathbf{V}_{diff} = \mathbf{V}_i - \mathbf{V}_{reset}$, which contains lots of consecutive zeros or ones. Compressing algorithms, such as RLE, would provide rather large compressing ratio on $\mathbf{V}_{diff}$. Knowing $\mathbf{V}_{diff}$ and $\mathbf{V}_{reset}$, we can recover $\mathbf{V}_i$ easily. Therefore, the compression-based recovery architecture allows us to memorize the system state with much fewer nonvolatile register, which means less silicon area.

III. Design Flow of Nonvolatile Processors

In this section, we present a general flow to transform a volatile processor into a nonvolatile one. Via the presented flow we evaluate the area and performance of a fabricated nonvolatile 8051 processor hereafter.

TABLE I
Normalized Area Utilization Before and After NVFF Replacement

Module Name		Before Replacement		After Replacement	
		Size	In Sum	Size	In Sum
MCU	MCU Controller	85.8	183.3	234.3	704.6
	ALU	16.0		16.0	
	UART	13.9		13.9	
	Timer/Counter	7.4		21.2	
	RAM	59.4		419.0	
SPI/I2C Controller	SPI Controller	6.9	23.0	6.9	23.0
	I2C Controller	16.1		16.1	
NVFF Controller		0	0	9.5	9.5
SRAM		381.0	381.0	381.0	381.0
JTAG		4.3	4.3	4.3	4.3
I/O Pad		1	1	1	1
Total Chip		597.0		1122.8	

A. Nonvolatile Design Flow

Given a volatile circuit, the nonvolatile design flow is shown in Figure 2. The first step is to determine the memory units in the circuit to be replaced by nonvolatile registers. Strictly speaking, all registers and RAMs should be replaced with nonvolatile components to store the system state completely, however the users may decide if any state can be omitted to save chip area. In the third step, we replace the memory units in the nonvolatile domain with nonvolatile registers. The replacement is deployed in the gate level. It can be executed in three steps: I) The RTL specification is synthesized into a gate netlist with Synopsys Design Compiler; II) As the nonvolatile registers are constructed from nonvolatile flip-flops (NVFF), we use an in house script to replace all volatile flip-flops with NVFFs; III) To make all NVFFs work in a proper sequence during the recovery process, a NVFF controller is integrated to instruct the behavior. Finally, we evaluate the nonvolatile design and decide if it meets the requirement. If not, we need more iterations from the beginning. Due to the page limit, we leave the details of the NVFF and controller design to our technical report [13].

B. Nonvolatile 8051 Processor with Full Replacement

According to the above design flow, we have taped out a 8051 nonvolatile processor based on a 0.13um ferroelectric process [13]. In order to retain the entire system state, all registers and 128 byte internal RAM are replaced with NVFFs. The total number of NVFFs reaches 1607. The SRAM is not replaced because SRAM don't memorize system state in target applications. To evaluate the area utilization, we give out the normalized area of each component in Table I. As we can see, the full replacement strategy increases the chip area by nearly 90%. To meet the area efficient design metric, we propose the compression-based recovery architecture instead of the full replacement solution in Section IV.

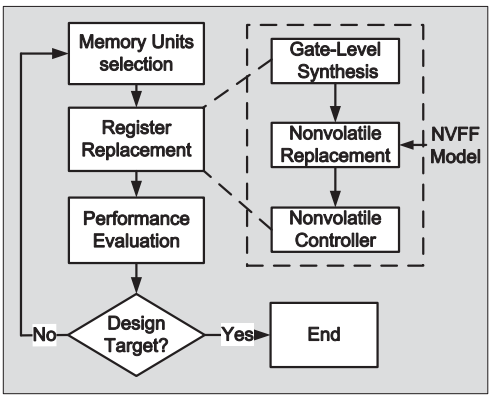

Fig. 2. Design Flow from a Volatile Processor to a Nonvolatile One

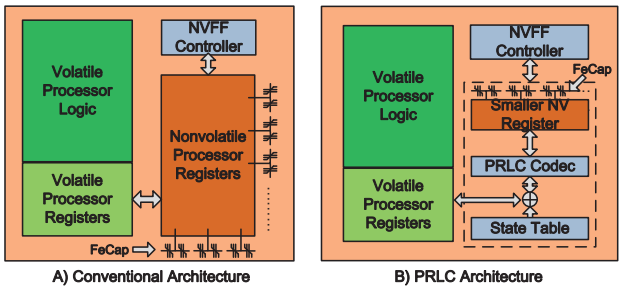

Fig. 3. Conventional Architecture vs PRLC Architecture

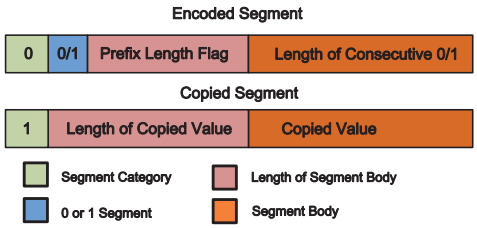

Fig. 4. Encoded and Copied Segment Structure

IV. Area Efficient Recovery Architecture

This section proposes a parallel run-length codec (PRLC) architecture to reduce the number of nonvolatile registers. It consists of an improved PRLC algorithm and the corresponding hardware implementation. Furthermore, we present a state table selection solution which strongly influences the PRLC's compression ratio.

A. PRLC Architecture

We compare the conventional nonvolatile processor with the proposed PRLC architecture in Figure 3. The PRLC architecture compresses the state vector $\mathbf{V} = (v_1, v_2, ..., v_n), v_i = 0/1$ and stores the compressed data in the nonvolatile registers, while the traditional processor stores $\mathbf{V}$ directly. In PRLC, the encoding procedure is listed as followings:

- Fetch the state vector $\mathbf{V}_{org}$ from the processor,
- Select a reference vector $\mathbf{V}_{ref}$ from the state table,
- XOR $\mathbf{V}_{org}$ and $\mathbf{V}_{ref}$ and get the differential vector $\mathbf{V}_{diff}$,
- Compress $\mathbf{V}_{diff}$ and get the compressed vector $\mathbf{V}_{comp}$.

The decoding procedure works in the opposite direction. Therefore, the reduced NVFF number $NVFF_{rd}$ is determined by equation 1:

$$NVFF_{rd} = |\mathbf{V}_{diff}| - |\mathbf{V}_{comp}| \quad (1)$$

Obviously, $NVFF_{rd}$ is correlated with the RLC algorithm and the distribution of zeros and ones in $\mathbf{V}_{diff}$. Thus, the PRLC design and the vector selection of the state table are critical. We will discuss those two points in the following.

B. Threshold Based RLC Algorithm

To achieve good compression ratio, we threshold based algorithm to record the length of consecutive 0 and 1. A threshold is selected to deal with short zeros and ones sequences. The threshold based variable run-length encoding (RLE) algorithm is described in Algorithm 1; The input of the algorithm is the system state vector $\mathbf{V}_{diff}$ and specified threshold m, the output is the compressed vector $\mathbf{V}_{comp}$. Line 1 checks if the encoding is complete or not by comparing vector current index k with the vector bound n. Line 2 determines if a transition ($0 \Rightarrow 1$ or $1 \Rightarrow 0$) happens in the sequence. If the number of the consecutive zeros or ones are smaller than m, $Process\_ShortSEQ$ is called to copy the segment into $\mathbf{V}_{comp}$.

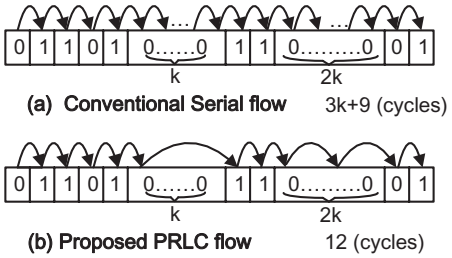

Fig. 5. Conventional and PRLE Encoding Flow

Otherwise, we call $Process\_LongSEQ$ to encode the segment into $\mathbf{V}_{comp}$. The encoded segments and copied segments are regularly

Algorithm 1 Threshold Based Variable RLE Algorithm

Input: $\mathbf{V}_{diff}, m$
Output: $\mathbf{V}_{comp}$
1: **while** $k < n$ **do**
2: **if** $a_k \neq a_{k+1}$ **then**
3: **if** $counter \leq m$ **then**
4: $Process\_ShortSEQ()$;
5: **end if**
6: **if** $counter > m$ **then**
7: $Process\_LongSEQ()$;
8: **end if**
9: **end if**
10: **end while**

constructed according to Figure 4. As shown in Figure 4, the first bit (flag bit) indicates the segment's category. The following 4 bit (length part) indicates the length of body part. The body part records the length of consecutive 0/1 for the encoded segments or the copied segments. The implementation of the decoder is straightforward. To achieve optimal compression ratio, the threshold value m should be properly chosen. In real applications, we found the optimal m varies quite small for different input vectors. The value of m is set to balance the length of the encoded segments and the copied segments. In our experiments, $m = 8$ for consecutive 0 and $m = 7$ for consecutive 1.

C. PRLC Implementation

Traditional serial RLE processes input vectors bit by bit. It deteriorates the data store and restore performance greatly. Trein et.al. [14] proposed a run-length encoding architecture with parallel inputs. However, their structure targeted at the image applications and considered the encoder only. In this paper, we develop an area-efficient PRLC architecture to accelerate the vector codec. The Parallelism means we can process consecutive 0 or 1 in one cycle. Figure 5 shows an ideal case to process a $(3k + 9)$-bit vector. The proposed method only needs 12 cycles to encode the vector. What's more, the larger the k becomes, the better speedup the method receives.

1) System Diagram: Figure 6 shows the system diagram. It consists of three parts: the parallel input, the PRLC block, and the parallel output. The parallel input contains a m-bit volatile register array and a i-bit barrel shift network. Similarly, the output block contains a n-bit nonvolatile register array and j-bit barrel shift network. The PRLC block realized the encoding in Figure 5 and the decoding process. For the input block, the user can configure the barrel shift network to transfer i-bit or less data between the register array and the PRLC buffer in parallel. The encoding flow of architecture is described as follows:

- Compare $\mathbf{V}_{org}$ with $\mathbf{V}_{ref}$ in parallel and write back $\mathbf{V}_{diff}$ to the m-bit register array,
- PRLC block judges if the current k-bit($k \leq i$) inputs are all zeroes and controls the i-bit barrel shift network to move k bits or 1 bit, and

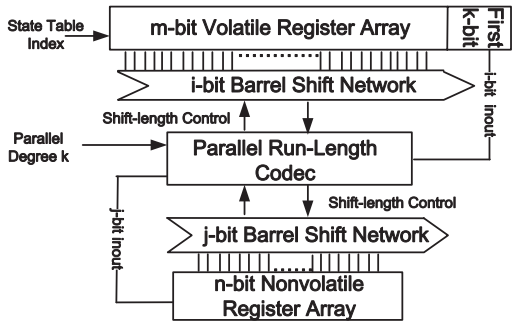

Fig. 6. The Proposed PRLC System Diagram

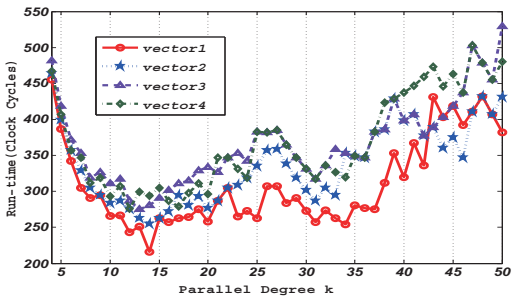

Fig. 7. Encoding Run-time under Different Parallel Degree

- After k cycles or 1 cycle, the PRLC generates one segment and controls the j-bit barrel shift network to write the segment into the compressed register vector.

The decoding process are listed as below:

- The PRLC reads in one segment,
- The PRLC shifts out the decoded vector q-bit by q-bit to the m-bit register array($q \leq i$), and
- Compare the decoded vector $\mathbf{V}_{diff}$ with $\mathbf{V}_{ref}$ and write back to the m-bit register array.

In this architecture, a proper k (q) affects the system performance significantly. Figure 7 shows the encoding clock cycle number under different parallel degree k with different input vectors. There is a k to achieve the smallest number of clock cycles. It is because a higher k may lead to more cycle reduction but may also lower the possibility of encountering consecutive zeros. Our experimental results show that though the optimal k may vary under different vectors. However, it usually locates in a fixed range $10-15$. Furthermore, our barrel shifting architecture also supports dynamical tuning k by software. Similarly, we can find a proper value for q in the decoding process. The proper q is i for the encoded segment. In other cases, it equals to the length of the copied segment.

2) Area and Performance Model: We give the area and performance models of the PRLC in this section. The PRLC architecture includes two barrel shift networks, a state table, a PRLC codec and an XOR array. We denoted the area of these modules in Table II. The area of the barrel shift network and the XOR array can be calculated by multiplying the number of cells with its unit area. A x-bit barrel shift network contains $xlog_2(x)$ multiplexers [15]. The area of a multiplex or a XOR gate can be obtained from the gate level estimation. The PRLC codec module is synthesized under Synopsys Design Compiler. The state table consists of several MOS switches which can be ignored. Thus, the total area reduction A_{rd} using the PRLC architecture can be expressed as:

$$\begin{aligned} A_{rd} &= (m-n)\times A_{nvff} - m\times A_{ff} \\ &\quad -(ilog_2(i)+jlog_2(j))\times A_{mux} \\ &\quad -m\times A_{xor} - A_{prlc} \end{aligned} \tag{2}$$

The running cycles in the encoding and decoding process can be expressed as follows(k,q is defined as above):

$$N_{EncodingCycle} = \sum_{e=1}^{E}\lceil\frac{L_e}{k}\rceil\times(k-1)+\sum_{c=1}^{C}L_c \tag{3}$$

$$N_{DecodingCycle} = \sum_{e=1}^{E}\lceil\frac{L_e}{q}\rceil + C \tag{4}$$

where E and C denote the number of the encoded and copied segments. L_e and L_c represent the length of each encoded and copied segment.

TABLE II
AREA DENOTATION FOR CRITICAL CELLS

Standard Cell	NV DFF	CMOS DFF	2-input XOR	1to2MUX	PRLE codec
Area Denotation	A_{nvff}	A_{ff}	A_{xor}	A_{mux}	A_{prlc}

D. State Table Optimization

We first discuss the architecture of the state table and then propose an algorithm to obtain a vector $\mathbf{V}_{ref}$ to maximize the compression ratio.

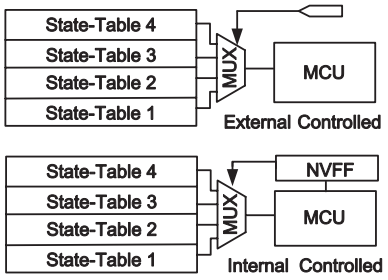

Fig. 8. Two Proposed State-Table Structure

1) State Table Architecture: The compression ratio strongly depends on the reference vector $\mathbf{V}_{ref}$ because it impacts the consecutive zeroes in the differential vector $\mathbf{V}_{diff}$. In different applications, the optimal $\mathbf{V}_{ref}$ are geratly different. Therefore we develop a nonvolatile reconfigurable state table architecture containing multiple reference vectors to support different applications. Figure 8 shows the state table, consisting of a table with multiple reference vectors and a selecting multiplexer. Each $\mathbf{V}_{ref}$ is optimized for one application or similar applications, which will be executed in an embedded system(e.g. wireless sensor networks). The state table can be implemented with a few MOS switches and interconnects with ignorable area. It is fixed and nonvolatile. Furthermore, we need also record the selection of the reference vector to restore the original system state. We proposed two reconfigurable solutions as below: One method adopts external inputs from nonvolatile mechanical switches or other control chips. The other uses several-bit NVFFs (e.g.2-4 bits) to memorize the selection. We may also use a hybrid strategy to combine these two methods.

2) Reference Vector Selecting Algorithm: In this part we present an algorithm to find optimal vectors to maximize the compression ratio under a specific application. We formulate the problem as below: Assuming β system state vectors $\{\mathbf{V}_1,\mathbf{V}_2,...,\mathbf{V}_\beta\}$ should be stored, we need determine an optimal reference vector $\mathbf{V}_{opt}$ to minimize the length $L_{\mathbf{V}_{comp}}$ of the compressed vector under the worst case:

$$L_{\mathbf{V}_{comp}}(\mathbf{V}) = \max_{i=1}^{\beta} P(\mathbf{V}_i \oplus \mathbf{V}) \rightarrow \min \tag{5}$$

TABLE III
NORMALIZED AREA OF STANDARD CELLS IN A TYPICAL MANUFACTURING PROCESS

$StandardCell$	NV DFF	CMOS DFF	2-input XOR	1to2MUX
$Area$	28.56	5.53	1.00	1.47

,where $\mathbf{V}_i \oplus \mathbf{V}$ represents XOR two vectors bit-by-bit; Function P calculates the length of the vector after compressing $\mathbf{V}$ by PRLC. The exploring space is $2^{|\mathbf{V}|}$, which is a NP problem. Therefore, we develop a heuristic method to find a solution. Intuitively, we synthesize a suboptimal reference vector $\mathbf{V}_{sub}$, which should have the least difference from other vectors. We set the ith bit of $\mathbf{V}_{sub}$ as follows:

$$\mathbf{V}_{sub}(i) = M(\{j \in 1, 2, ..., \beta | \mathbf{V}_j(i)\}) \quad (6)$$

$M(\mathbf{S})$ equals to the majority element in the set $\mathbf{S}$. This method can achieve quite good compression ratio in most cases, however it may lead to poor results for some special vectors. We will discuss those situations in the next section.

3) Discussion on Worst Cases: If the value of $L_{\mathbf{V}_{comp}}$ is larger than the number of real nonvolatile register array, we call that an overflow occurs. In that case, the system state can not be stored correctly. However, it is both expensive and unnecessary to enumerate all possible lengths of $\mathbf{V}_{comp}$ in various applications, so as to ensure no overflows happen. Alternatively, we adopted a length of nonvolatile resgisters to keep the possibility of the overflow under a threshold. Moreover, we also design an error-discarding mechanism to handle the worst case. If an overflow occurs, we keep the previous state system untouched. Therefore, the processor will roll back to its previous state instead of the current state. Although it causes performance loss, it is acceptable when overflows rarely happen. We demonstrate the effectiveness of this approach in Section V.

V. EVALUATION

We first explain the experimental configurations. After that, we compare the traditional fully-replaced architecture with the PRLC by area and performance. Furthermore, we analyze the possibility and distribution of the overflows. Finally, we evaluate the state table selecting algorithm under different benchmarks. The data used in evaluation is based on our taped-out nonvolatile processor.

A. Experimental Configuration

In the experiments, we evaluate the PRLC architecture in a 8051-compatible processor. We use Cadence NCVerilog to obtain the system state vector. We estimate the area of the original nonvolatile process with Synopsys Design Compiler under Rohm's $0.13\mu m$ ferroelectric process. The normalized area of the standard cell is shown in Table III. To achieve a better performance, we set $i = 16$ and $j = 21$. The benchmarks used in the experiments are listed as below:

- **FFT**:8-bit fast fourier transformation,
- **AES**:a symmetric key encrypt algorithm,
- **Bubble Sort**:a popular sorting algorithm,
- **KMP** Searching:a string matching algorithm,
- **Matrix Multiply**:8x8 matrix multiplying operation,and
- **ZigBee Protocol**:ZigBee MAC protocol implementation.

B. PRLC Evaluation

We first evaluate the area efficiency of the PRLC architecture under different benchmarks. We randomly pick up 50 original vectors in each benchmark and calculate the reference vector $\mathbf{V}_{sub}$ based on Equation 6. According to Equation 5, we adopt the largest length of the compressed vectors in the samples as the number of the nonvolatile register array. We get the area efficiency results in Table IV. Given a fixed reference vector $\mathbf{V}_{sub}$, the system state under different applications may lead to compressed vectors with different length. Obviously, the area savings are variable for different benchmarks. We achieve up to 43.2% savings for the MCU only case and 28.6%(the best case) for the MCU with 8 KB SRAM with the average rating range from 25% to 38%. The number the nonvolatile register is reduced from 1607 to 453(the worst case). The possibility of the overflow is under 1.5%.

TABLE IV
EVALUATION OF AREA EFFICIENCY OF PRLE ARCHITECTURE

Benchmark	Compression Ratio	# of NV registers	Area Reduction Ratio		Overflow Possibility
			MCU only	MCU+8KB SRAM	
FFT	22.7%	365	40.0%	26.5%	0.6%
AES	18.5%	298	43.2%	28.6%	0.8%
Bubble sort	22.9%	369	40.0%	26.4%	0.9%
KMP Searching	26.8%	430	36.9%	24.5%	0.9%
Matrix Multiply	26.1%	420	37.4%	24.8%	1.5%
MAC protocol	28.2%	453	35.8%	23.7%	0.5%

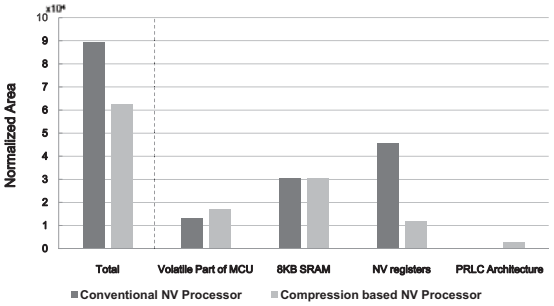

Fig. 9. Normalized Area allocation in Conventional NV-Processor and PRLE Solution under Different Benchmarks

We show the normalized area breakdown of the traditional nonvolatile processor and the proposed one in Figure 9. The nonvolatile registers contribute to more than 50% percent without the PRLC architecture, while the percent of NVFFs is reduced to less than 20% in the proposed architecture. Furthermore, the PRLC occupies less than 4% percent silicon area, which is is trivial compared with the large savings in the nonvolatile registers. What's more, the saving ratio could be further increased with smaller SRAMs.

Table V shows the clock cycles of encoding and decoding under different benchmarks. For each benchmark, we first decide the optimal parallel degree k by simulations and then calculate the clock cycles. Our results show that the optimal k ranges from 10 to 16 under different applications. It enables us to use a universal k-bit barrel shift network to support the dynamic tuning of the parallel degree. The encoding process need extra $200-300$ cycles to compress one vector, while the decoding one costs approximate 120 cycles. The total processing time is less than $50\mu s$ with a 10 MHz clock frequency. Considering the estimated back-up time in conventional NV-processor is $80\mu s$, the back-up time for PRLC based NV-processor is less than $130\mu s$, which is compatible to the instant on feature. Though the PRLC architecture leads to extra delays during the backup stage, it

TABLE V
EVALUATION OF RUN-TIME OF PRLE CODEC

Benchmark	Optimal k	Clock Cycles				Process Time on Average (μs)	
		Encode		Decode			
		Avg	Var	Avg	Var	Encode	Decode
FFT	14	189.6	27.1	115.4	5.6	19.0	11.5
AES	13	199.1	33.1	114.4	4.8	19.9	11.4
Bubble sort	15	245.3	39.7	114.2	7.3	24.5	11.4
KMP Searching	16	211.4	48.7	116.2	5.6	21.1	11.6
Matrix Multiply	13	211.3	25.0	116.4	4.2	21.1	11.6
MAC protocol	13	265.2	32.5	113.7	8.43	26.5	11.3

[1] Assuming the processor runs at 10MHz clock frequency

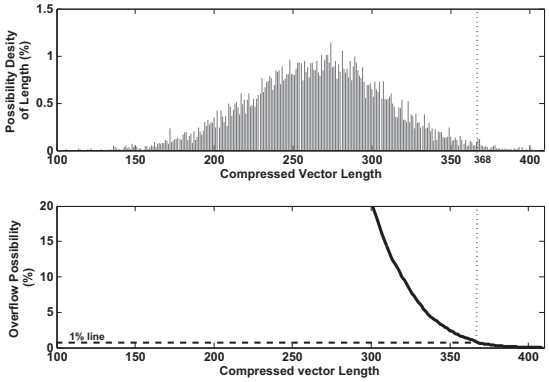

Fig. 10. Possibility Distribution and Compression Overflow Ratio of Different Length

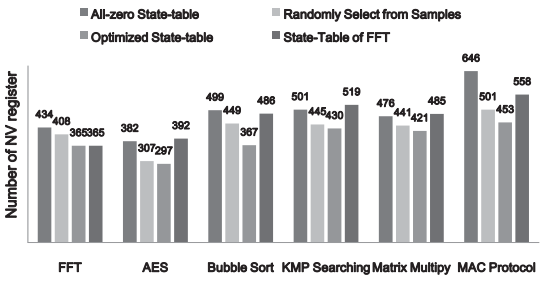

Fig. 11. State-table Impact on the Compressed Length of NVFF

is inactive and can be power gated in the normal mode. Thus, it doesn't cause extra power consumption and performance loss.

C. Overflow Analysis

We analyze the overflow possibility under different lengths of the nonvolatile register array. We samples the system state $\mathbf{V}$ in every clock cycle and calculate their length using PRLC. Figure 10 shows the possibility distribution of the compressed vector length $|\mathbf{V}_{comp}|$ with the Bubble Sort benchmark. (We observe similar phenomenons in other benchmarks.) It obeys a normal distribution and shows that the occurrence of a length above a certain threshold is quite rare. For this benchmark, we adopt a 368-bit nonvolatile register array to make the overflow possibility below 1%. It should be noted that this overflow possibility will happen if we store the system state cycle by cycle. In real applications, we usually store the system state in more coarse-grained granularity, which makes the possibility much lower.

D. Reference Vector Selection

We evaluate the compressed vector length $|\mathbf{V}_{comp}|$ under four vector selecting methods: all-zero reset vector, random vector, optimized vector by Equation 6, a system state vector from FFT . Figure 11 shows the results under different benchmarks. It is shown that different reference vectors can cause up to 42% variation of the compressed length. Choosing a system state vector from FFT works well for the FFT application, but it is not good for other applications. It implies that a application specific reconfigurable state table may work quite well. Furthermore, the proposed optimal vector works quite good under all benchmarks compared with the random and all-zero approach. It demonstrates that a universal reference vector can be selected for lots of applications, though it is not so good as the application specific vectors in some cases.

E. Discussion

Among existed mature nonvolatile processes, the hybrid structure in Figure 1 should be used due to the lifetime concern of nonvolatile components. In that case, nonvolatile register is always larger than the CMOS DFF symbolized as $S_{NVFF} > S_{DFF}$. Our techniques can reduce the chip area if $S_{NVFF} > S_{DFF}$. Therefore, it is applicable to most existed nonvolatile processes. However, it is unclear whether a nonvolatile register with smaller area than DFF can be mass produced in future.

VI. Conclusions

Nonvolatile processors, based on the emerging materials, open up a new domain for the power savings and attractive applications. However, the traditional fully-replaced design flow causes prohibitive area overheads and increases the manufacture costs. This paper proposes a state table based compression method to reduce the NVFF number. A novel PRLC architecture and a reconfigurable state table are implemented, and corresponding algorithms are developed to find the optimal parallel degree and the reference vector. Experimental results show that we achieve over 4 times NVFF reductions and up to 43% overall processor area savings. Meanwhile, only $50\mu s$ time overhead adapt it to both abrupt and planned power down.

Although we evaluate this method on a fabricated ferroelectric process based processor, it is also applicable to other designs with other nonvolatile materials based registers. Our future work aims at the integration of the PRLC architecture with our taped-out nonvolatile processor and evaluates the performance and area for more novel applications.

References

[1] Rohm Co., Ltd., "Rohm Demonstrates Nonvolatile CPU," *Website: http://techon.nikkeibp.co.jp/english/NEWS_EN/20071004/140206/.*
[2] W. Yu, S. Rajwade, S. Wang, B. Lian, G. Suh, and E. Kan, "a non-volatile microcontroller with integrated floating-gate transistors," in *Proceedings of the 5th Workshop on Dependable and Secure Nanocomputing*. ACM Press, 2011, pp. 1–4.
[3] C. Holland, "First MRAM-based FPGA taped-out," *Website: http://www.eetimes.com/General/DisplayPrintViewContent?contentItemId=4200035.*
[4] W. Zhao, E. Belhaire, V. Javerliac, C. Chappert, and B. Dieny, "Evaluation of a non-volatile fpga based on mram technology," in *Integrated Circuit Design and Technology, 2006. ICICDT'06. 2006 IEEE International Conference on*. IEEE, 2006, pp. 1–4.
[5] ITRS, "Roadmap for Nonvolatile Memory," *Website: http://www.itrs.net/.*
[6] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Design exploration of hybrid caches with disparate memory technologies," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, no. 3, p. 15, 2010.
[7] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai, "Nonvolatile magnetic flip-flop for standby-power-free socs," in *Custom Integrated Circuits Conference, 2008. CICC 2008. IEEE*. IEEE, pp. 355–358.
[8] W. Zhao, E. Belhaire, V. Javerliac, C. Chappert, and B. Dieny, "A non-volatile flip-flop in magnetic fpga chip," in *Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006. International Conference on*. IEEE, 2006, pp. 323–326.
[9] Nikkei Electronics Asia, "Rohm Develops Non-Volatile Register; Slashes Dissipation," *Website: http://techon.nikkeibp.co.jp/article /HONSHI/20080729/155646/.*
[10] N. Sakimura, T. Sugibayashi, R. Nebashi, and N. Kasai, "Nonvolatile magnetic flip-flop for standby-power-free socs," *Solid-State Circuits, IEEE Journal of*, vol. 44, no. 8, pp. 2244–2250, 2009.
[11] J. Wang, Y. Liu, H. Yang, and H. Wang, "A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications," in *Green Circuits and Systems (ICGCS), 2010 International Conference on*. IEEE, pp. 646–650.
[12] X. Guo, E. Ipek, and T. Soyata, "Resistive computation: avoiding the power wall with low-leakage, stt-mram based computing," in *2010 Proceedings of the 37th annual international symposium on Computer architecture*. ACM Press, 2010.
[13] anonymous, "Technical report," 2011.
[14] J. Trein, A. Schwarzbacher, B. Hoppe, and K. Noff, "A hardware implementation of a run length encoding compression algorithm with parallel inputs," in *Signals and Systems Conference, 2008.(ISSC 2008). IET Irish*. IET, pp. 337–342.
[15] P. Khandekar and S. Subbaraman, "Low power 2:1 mux for barrel shifter," in *Emerging Trends in Engineering and Technology, 2008. ICETET '08. First International Conference on*, july 2008, pp. 404 – 407.