Chapter 11
Ferroelectric Nonvolatile Processor Design, Optimization, and Application

Yongpan Liu, Huazhong Yang, Yiqun Wang, Cong Wang, Xiao Sheng, Shuangchen Li, Daming Zhang and Yinan Sun

Abstract Nonvolatile processor (NVP) is one of the most promising techniques to realize energy-efficient computing systems with zero standby power, instant-on features, high resilience to power failures, and fine-grained power management. As flip-flops as well as static random access memories (SRAM) should be replaced by nonvolatile memory in an NVP, it puts rigid requirements on the nonvolatile memories, such as nearly unlimited operation cycles, ultra-short access time and easy integration to CMOS technology. Ferroelectric memory can meet those metrics with good energy efficiency, which is appropriate to realize an NVP. However, there are several major design problems, such as the unknown design flow of a ferroelectric NVP, the nontrivial area overheads as well as the absent of the real application systems. To overcome those challenges, we present the first fabricated NVP with zero standby power, 7 μs sleep time and 3 μs wake-up time, consisting of a flip-flop controller (FFC), a distributed memory architecture and a voltage detection system. Compared with an existing industry processor, it can achieve over 30–100× speedup on the wake-up/sleep time and 70× energy savings on the data backup and recall operations. Meanwhile, the ferroelectric NVP exhibits comparative performance and power consumption in normal operations. To attack its area challenges, we design a parallel compare-and-compress architecture (PaCC) and an appropriate vector selecting method to reduce the number of nonvolatile registers by 70–80 % with less than 1 % overflow possibility, which leads to up to 30 % processor area savings. Another segment-based parallel compression (SPaC) architecture is proposed to trade off the chip area and the backup speed. It divides the system vector into several segments and compresses them in parallel. Compared with the PaCC solution, it can improve the backup speed by 83 % with 16 % area savings over the full replacement architecture. Finally, we demonstrate two kinds of battery-less sensor nodes based on the NVP for the first time.They aimed at the moving object detection and the body sensor appli-

Y. Liu (✉) · H. Yang · Y. Wang · C. Wang · X. Sheng · S. Li · D. Zhang · Y. Sun
Department of Electronic Engineering, Tsinghua University, Beijing, Peoples Republic of China
e-mail: ypliu@tsinghua.edu.cn

Y. Xie (ed.), *Emerging Memory Technologies*,
DOI: 10.1007/978-1-4419-9551-3_11,

cations. As both systems are powered by energy-harvesting systems, they eliminate the battery lifetime constraints and work reliably under frequency power failures.

11.1 Introduction

In embedded applications, low-power processors can be shut down to eliminate leakage power. However, their states in volatile storage elements, such as flip-flops, registers, and static random access memories (SRAM), will be lost, because the charge on the capacitors is quickly drained without power supply. To keep those states, they need to backup the states into the secondary nonvolatile storages, such as flash memory or hard disks. But transferring data between a volatile processor and secondary nonvolatile storages leads to long delays, large switching energy consumption, and vulnerability to power failures. Therefore, efficient alternatives are needed to remove such limitations.

Nonvolatile processor (NVP) is one of the most promising alternatives, which replaces flip-flops, registers, and SRAMs in processors with nonvolatile ones. As the nonvolatile memory can store the system states locally, data transfer overheads are significantly reduced. Researchers have evaluated various nonvolatile memories in processors [1–4] to show the following advantages: (1) zero standby power: the processor can retain its state without power, while the traditional ones suffer from the increasing leakage power to keep data; (2) instant-on and instant-off features: the processor can resume its work within several cycles from the stalled point, while the traditional one needs millions of initializing cycles; (3) high resilience to power failures: the processor can work reliably under the environments with frequent power interrupts, such as energy-harvesting and wireless-powered applications [2]; (4) 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 NVPs is interesting.

To implement an NVP, appropriate memory technologies are needed. Flash is a mature high-density nonvolatile memory in commercial microcontrollers [5, 6]. However, it has drawbacks of low endurance, slow writing speed, block erasing pattern, and high mask cost as distributed nonvolatile registers. Phase-change random access memory (PCRAM) has the potential to be used as a cache with SRAMs [7]. However, its asymmetric reading/writing characteristics and limited lifetime hinder its application as flip-flops and registers. Among existing nonvolatile memories [8], ferroelectric RAM (FeRAM) and magnetic RAM (MRAM) emerge as the most promising candidates for NVPs because of their nearly unlimited operation cycles, ultra-short access time and easy integration to CMOS technology. As FeRAM shows better energy efficiency and mature fabrication process, this chapter will focus on the ferroelectric NVP.

There are several major design challenges from the ferroelectric NVP. First, though previous work had reported the implementation of nonvolatile flip-flops (NVFF) [9–11], the complete design flow to implement a ferroelectric NVP is

still absent. Besides NVFFs and RAMs, it is necessary to build flip-flop controllers (FFCs), distributed memory architecture, peripheral power manage circuits, and specific design tools. However, none of previous work solved those problems. Second, as most of popular NVFFs adopt hybrid architecture, researchers observed nontrivial area overheads. In [2, 10, 12], they reported over 40 % increasing of chip area due to the replacement of nonvolatile registers. It will have a great impact on the cost and yield. Therefore, investigating techniques to realize area-efficient NVPs is necessary. Finally, the absent of a real NVP hinders the exploration of its application. There are no reports on how to use NVPs and combine their characteristics with appropriate applications [13–15], such as smart sensors and energy-harvesting devices. To overcome those challenges from NVP design, optimization, and applications, our contributions are listed as below:

1. We present the first fabricated NVP with zero standby power, 7 μs sleep time, and 3 μs wake-up time, consisting of FFC, distributed memory architecture and voltage detection system.
2. We demonstrate that the NVP can achieve over 30–100× speedups on the wake-up/sleep time and 70× energy savings on the data backup and recall operations compared with an existing industry processor. Meanwhile, the NVP exhibits comparative performance and power consumption in normal operations.
3. We propose a parallel compare-and-compress architecture (PaCC) to reduce the area of nonvolatile registers. With an appropriate vector selecting method, the compression codec can reduce the number of nonvolatile registers by 70–80 % with less than 1 % overflow possibility, which leads to up to 30 % processor area savings.
4. We present a SPaC architecture to trade off the chip area and the backup speed. With a hybrid off-line/online partition method, it divides the system vector into several segments and compresses them in parallel. Compared with the PaCC solution, it can improve the backup speed by 83 % with 16 % area savings over the full replacement architecture.
5. We demonstrate two kinds of battery-less sensor nodes based on the NVP. They aimed at moving vehicle detection and body sensor applications. As both systems are powered by energy-harvesting devices, they eliminate the battery lifetime constraints and work reliably under frequency power failures.

The chapter is organized as follows: Sect. 11.2 discusses the mechanism of ferroelectric memory and the models. We introduce the design of NVP in Sect. 11.3 and present the related area optimizing techniques in Sect. 11.4. We discuss two applications of NVP in Sect. 11.5 and present the related work in Sect. 11.6. Section 11.7 concludes the chapter.

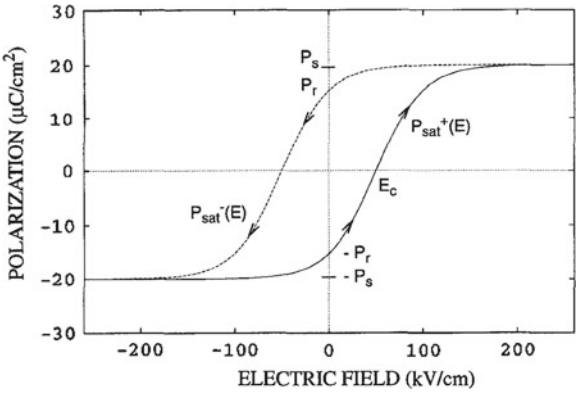


Fig. 11.1 Polarization–voltage hysteretic loop [16]

11.2 Background

This section describes the mechanism and the model of ferroelectric capacitor (FeCap) in the memory cells. It is an indispensable background for the ferroelectric NVP design.

11.2.1 Ferroelectric Material Mechanism

A ferroelectric material refers to a material with two stable polarization states. It switches from one state to another if an external strong electric field is applied. Furthermore, the relationship between the electric field E and the electric displacement D of a ferroelectric material is hysteretic. Figure 11.1 shows a typical polarization–voltage (P–V) hysteretic loop. After removing the external electric field, the remanent polarization $+/-P_{\mathrm{r}}$ represents logic "0" or "1". Those polarization information can be stored reliably without power supply.

With the stable polarization states of ferroelectric materials, we can store information in FeCaps by applying an external field. Figure 11.2 shows the popular designs of a ferroelectric memory cell, including the 1T-1C and the 2T-2C architecture. The 1T-1C design uses one FeCap to store the information, while there is a complementary FeCap pair in the 2T-2C architecture [17]. When we need to read a specific FeCap in a 1T-1C cell, a poled reference cell is used. By measuring the voltage/current between the capacitor and the reference cell, we can determine the stored value. The 1T-1C cell is area efficient but less reliable. The 2T-2C cell uses two FeCaps to store complementary polarization states. It can store and recall the information more reliably but with much larger area overheads.

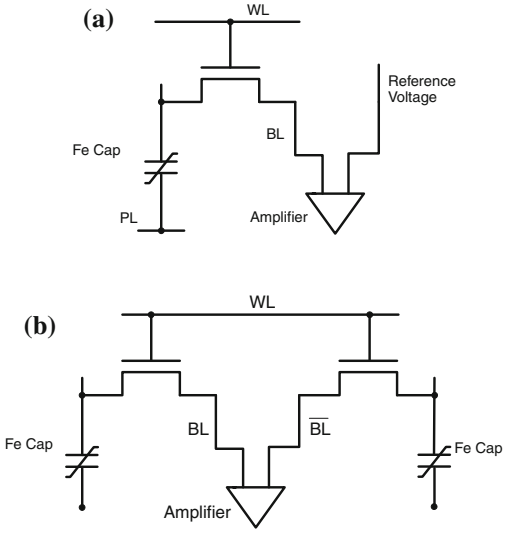


Fig. 11.2 Memory design with FeCaps [17]. **a** 1T1C, **b** 2T2C

11.2.2 Ferroelectric Capacitor Simulation Model

To better design and optimize the ferroelectric memory, we need a circuit model to efficiently describe the electrical characteristics of a FeCap. There are several models for FeCaps, such as zero-switching-time macro-modeling [16, 18]. However, they are computationally expensive. Furthermore, previous models generally omitted to model the capacitor during the power-off state, which is desirable for the unified simulation framework.

For circuit design, we need a behavior model for the SPICE simulator and characterize the nonvolatile property when power failures occur. Therefore, we built the FeCap model in Fig. 11.3. It consists of two nonlinear voltage-control capacitors, a voltage-control voltage source, and some voltage-control switches. A Schmitt trigger generates the control signals for switches S1–S4. Two voltage-control resistances mimic the nonvolatile property. Their values are zero when the power is on and tend to be infinite when the power is off to keep the charge in the capacitors.

This model can be easily implemented in SPICE, and the simulation results of P–V hysteresis loop are shown in Fig. 11.4. By tuning the circuit parameters of the model, the curve coincides with the measured data. The behavior model of ferroelectric capacitors can be used in the SPICE simulation for the NVFF design.

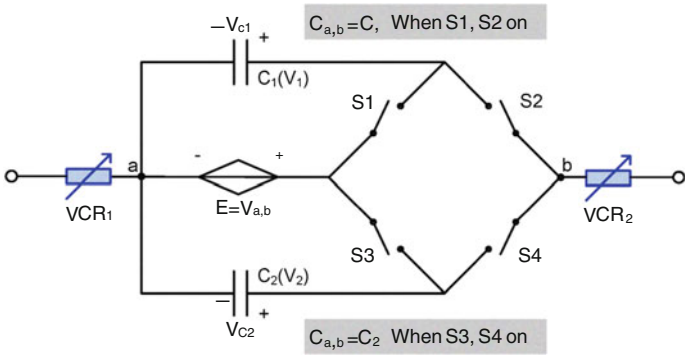


Fig. 11.3 Ferroelectric capacitance SPICE model

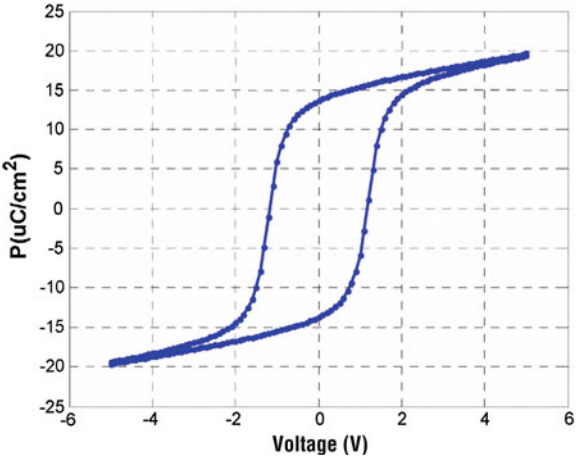


Fig. 11.4 Simulation result of P–V hysteresis loop

11.3 Nonvolatile Processor Design

In this section, we will introduce an actual NVP design based on ferroelectric flip-flops. First, we show the overall architecture of our design. Second, we describe two special nonvolatile circuits differing from a volatile processor: the ferroelectric-capacitor-based flip-flops and the FFC. Third, we discuss how NVP works under power failures. Finally, we give out some measured results from the fabricated NVP.

11.3.1 Overall Architecture

Figure 11.5 shows the block diagram of the fabricated NVP. It contains an MCS51 instruction set compatible core, a mode-selecting module, a JTAG debugging module,

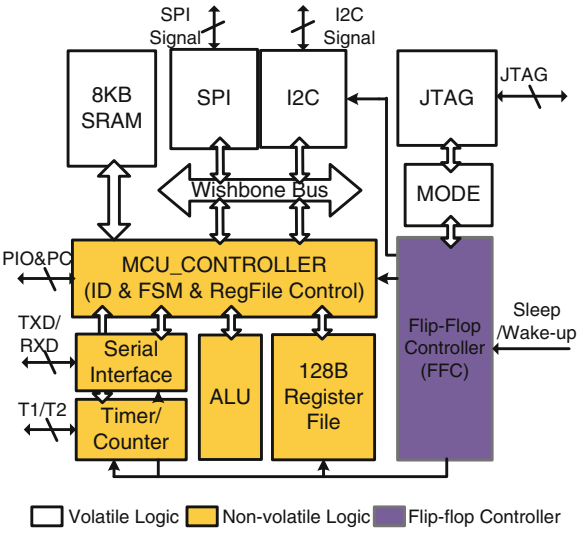


Fig. 11.5 Overall architecture of NVP

and two peripheral interfaces. The core consists of an MCU controller, an 8-bit arithmetic logic unit, a serial interface, a 128-byte register file, and an 8 K-byte SRAM. We have replaced the 128-byte register file and all flip-flops in the core with FeCap-based flip-flops (FeFFs). As the SRAM is used as a secondary data memory, we do not replace it in this implementation. However, it can be replaced with FeRAM easily [19]. The mode-selecting module controls the operating mode of the processor, including the volatile mode, the nonvolatile mode, and the debug mode. The JTAG module supports the user to debug the processor via scan chains. The SPI and I2C interfaces are connected to the core via a Wishbone bus. They are realized in the volatile logic domain. The FFC provides controlling signals to all FeFFs and volatile flip-flops in parallel after receiving the sleep or wake-up signals. As the FeFF and FFC are two main parts differing from a volatile processor, their detail structures are described as follows.

11.3.2 FeFF Design and Optimization

Figure 11.6a [12] shows the FeFF diagram. It adopts a hybrid CMOS and ferroelectric process, consisting of a standard master-slave D flip-flop (DFF) and a backup FeCap pair. They are isolated by two CMOS switches M1/M2 controlled by the "RW" signal. In the normal operating mode, the switches M1/M2 are open so the FeFF works as a standard DFF. Therefore, no performance losts will be introduced to the NVP during the normal operations. When a sleep/wake-up signal is detected, the

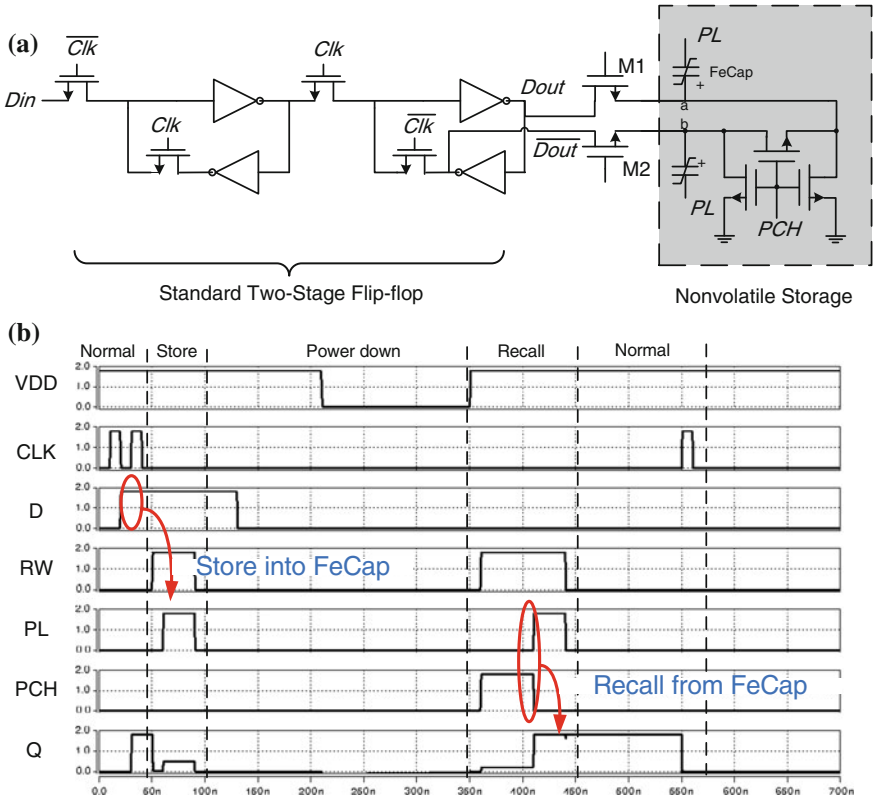


Fig. 11.6 Modeling and simulation of FeFF. **a** Proposed ferroelectric flip-flop structure. **b** Timing char of store and recall

controller will generate signals to make FeFFs store and restore their states and close the switches M1/M2. In the store operation, the complementary outputs of the DFF are connected to the positive plates of the FeCaps. "PL" is pulled up to polarize the FeCap pair into different complementary values to remember the present state. In the restore operation, "PCH" is pulled up to shorten nodes "a" and "b". The back-to-back inverter pair operates in a semi-stable state. After that, "PL" is pulled up and "PCH" is pulled down simultaneously. The FeCap pair drives nodes "a" and "b" with different currents until nodes "a" and "b" stay at stable complementary outputs "0/1" or "1/0". The differential architecture improves the reliability and performance with area overheads. We simulate this circuit in HSPICE, and Fig. 11.6b shows the waveform of the circuit during store and restore operations.

11.3.3 Flip-Flop Controller

The FFC generates controlling signals to FeFFs during sleep and wake-up actions. Figure 11.7a shows the block diagram of a FFC. It consists of a timing block and a finite-state machine (FSM). The timing block is self-timed by the inverter chain and the three timers provide overflow signals (Tov1-Tov3) to the FSM. The FSM generates the controlling signals ("RW," "PL," "PCH") based on Tov1–Tov3 to meet the timing requirements in Fig. 11.7c. The "Sleep/Wake-up" signal triggers the FSM to execute the sleep sequence or the wake-up sequence. The output "CG" is the clock-gating signal to both FeFFs and DFFs. Figure 11.7b shows the interconnections between the FFC and the flip-flops. "CG" gates the clock of both the FeFFs and the volatile flip-flops during the store and recall actions in Fig. 11.7c. It prevents writing uncertainty when the FeFFs executing store operations and halts the system before the FeFFs complete the data recall.

11.3.4 Power Management Circuit and Optimization

This section will discuss how to generate the sleep/wake-up signal for the NVP to backup its system state when power failures happen. An configurable voltage detection system (CVDS) is proposed to generate those signals.

The CVDS architecture is shown in Fig. 11.8a, which contains two configurable units. The first one is a switched capacitor array attached to the power line. The configurable capacitor, denoted as C_{PL}, provides the data backup energy to the NVP by

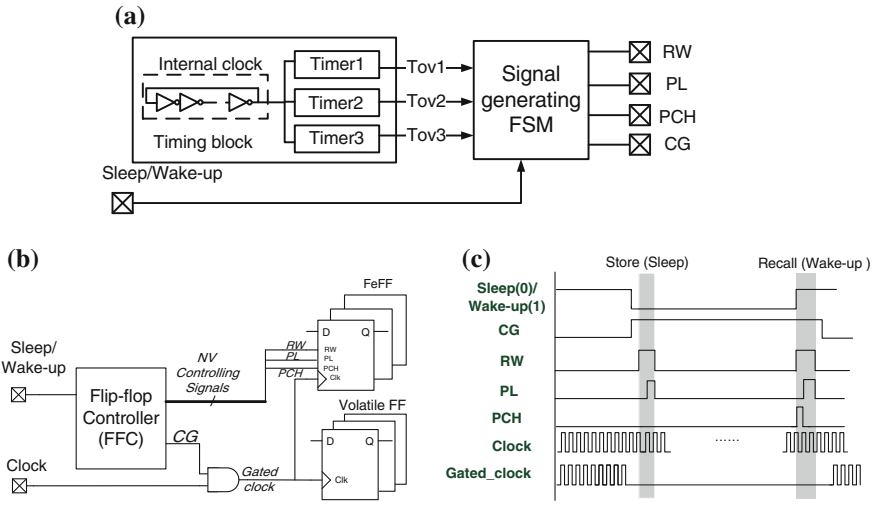


Fig. 11.7 Block diagram of flip-flop controller. **a** Block diagram of FFC. **b** Interconnection between FFC and FFs. **c** Timing chart in store and recall actions

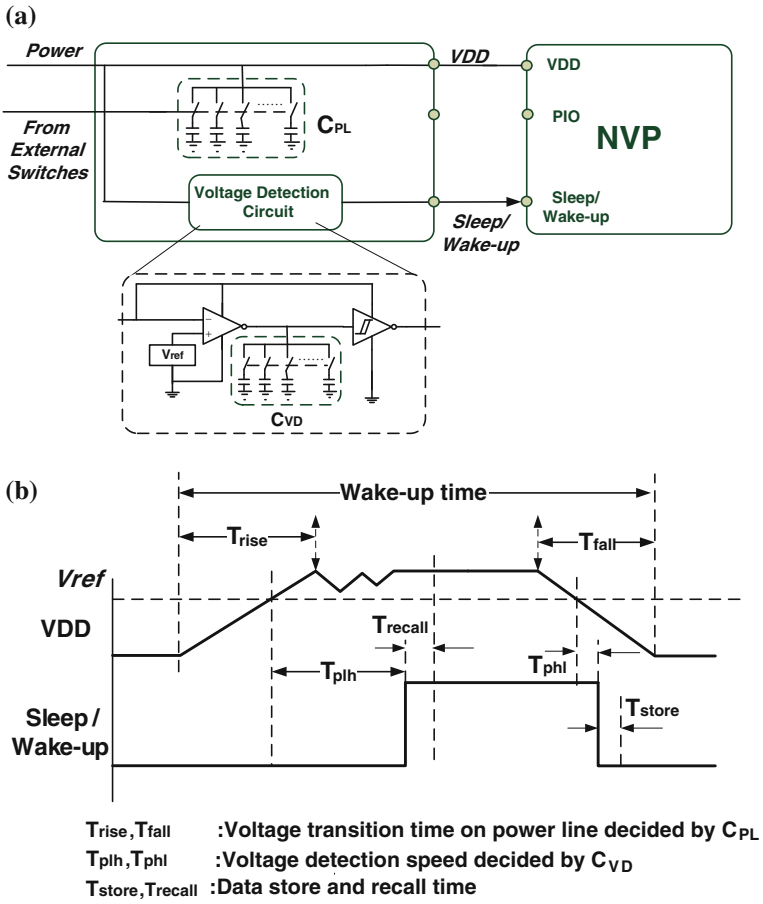


Fig. 11.8 Architecture and controlling timing chart of configurable voltage detection system. **a** Architecture of configurable voltage detection system. **b** Timing chart of system level sleep/wake-up process

keeping the voltage above the operating threshold after power down. The other one, denoted as C_{VD}, is another switched capacitor array used in the voltage detection circuit. The voltage detection circuit detects the power failure and recovery. It generates the sleep/wake-up signal (0/1) within a specific time (T_{plh}) after the voltage of the power line is above the threshold. We set the value of T_{plh} considering both the system reliability and the backup speed. The controlling words for those switched capacitor arrays are given by external input switches. Figure 11.8b shows the signal waveform during the sleep/wake-up actions and the timing diagram influenced by C_{PL} and C_{VD}.

We show the measured results of the wake-up time with different capacitance of C_{PL} and C_{VD} under different power conditions in Fig. 11.9. The wake-up time

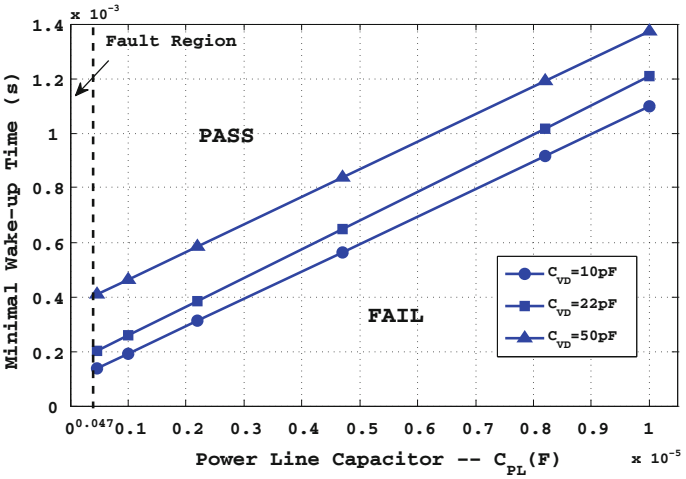


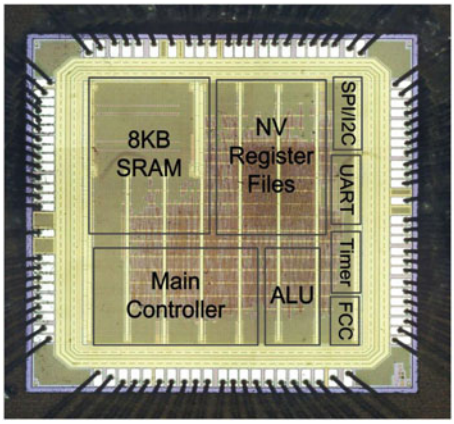
Fig. 11.9 The minimal wake-up time for system accuracy under different capacitance values

should be larger than the sum of power-stabilizing time, voltage detection time, and on-chip intrinsic wake-up time. C_{PL} has an impact on the power-stabilizing time via the RC-constant, where the minimal requested wake-up time decreases with smaller C_{PL}. However, when C_{PL} is smaller than 470 nF, it cannot provide sufficient energy for the NVP to backup the states so that the system goes to the false region, where the NVP can not backup its states correctly. C_{VD} affects the voltage detection time via its RC-constant, where the wake-up time decreases with smaller C_{VD}. C_{VD} cannot be smaller than 10 pF for the power and clock signals to be stabilized. As we can see, the minimal system wake-up time ($>100\,\mu$s) is much larger than the on-chip intrinsic sleep and wake-up time ($<10\,\mu$s), because the most dominating factor is the time for the power and clock signals to be stabilized, instead of the on-chip intrinsic circuit delay.

11.3.5 Chip Testing

The fabricated NVP, named as **THU1010N**, uses the ROHM 0.13 µm CMOS-ferroelectric hybrid process. Figure 11.10 shows its photomicrograph and general statistics. It has two unique nonvolatile characteristics: 1,607-bit FeFFs and zero standby power. The maximum operating frequency and the active power consumption are measured under a suite of embedded benchmarks for sensor networks, including FFT, FIR, and ZigBee protocols. Specially, we show the properties of sleep and wake-up actions, as well as some comparison results in Table 11.1.

Table 11.1 compares the NVP with a popular industrial processor "MSP430" [5] and its variants with FeRAM "MSP430FR series" [19]. We show the sleep/wake-up



Die Area	3.4mm×3.4mm (Pad Dominated)
Pin Number	100
Max Clock Feq.	25MHz
Core Voltage	1.2V
Capacity of Memory	1607-**bit FeFF** 8KB SRAM
Nonvolatile Level	Register level
Active Power	106uW/MHz
Standby Power	0

Fig. 11.10 Micrograph and general design statistics of THU1010N

Table 11.1 Comparison results of sleep and wake-up properties between THU-1010N and emerging commercial microprocessor

Microprocessor type		THU1010N	TI-MSP430-5 series with Flash [5]	TI-MSP430 with FRAM [19]
Sleep/Wake-up time	Sleep time (μs)	7	6×10^3	212
	Wake-up time (μs)	3	3×10^3	310
Sleep/Wake-up energy	Sleep energy (pJ/bit)	14.4	2.76×10^5	N/A
	Wake-up energy (pJ/bit)	5.04	373	N/A

time and the energy consumption during the store and recall operations. The sleep and wake-up time of "MSP430" come from the switching time between LPM4.5 mode and active mode [5, 19]. The results show that the NVP has tremendous advantages in the sleep/wake-up time and energy. It achieves over 100–1, 000× speedup in the sleep time, 30–100× speedup in the wake-up time, and 19, 000× saving in the sleep energy. In the chip level, we need a few microseconds to switch from the power-down mode to the active mode. Therefore, we conclude that the distributed nonvolatile architecture of the NVP provides much better performance than the existing centralized nonvolatile storage. It is quite promising for the energy-harvesting and power management applications to use the NVP technique.

Moreover, we discover that the core voltage has an effect on the sleep/wake-up speed. Figure 11.11 shows the sleep/wake-up time under different supply voltages. Lower voltages lead to a slower speed. It is because the delays of both FeFFs and the FFC circuit become larger under a lower voltage. However, the sleep/wake-up time can still be smaller than 20 and 10 μs under a 0.8-V supply voltage. It implies that the NVP can keep its instant-on feature under low voltages. Compared with the several milliseconds backup time in the centralized structure, the microseconds switching time will provide more power saving opportunities for the fine-grained on-chip power management.

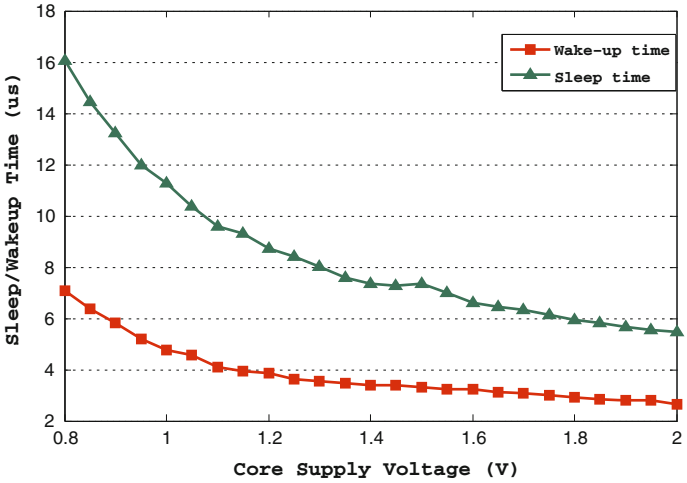


Fig. 11.11 Sleep and wake-up time in an NVP under different supply voltages

11.4 Compression Codec for Nonvolatile Processor

Though the NVP can sleep and wake-up very quickly, the nonvolatile storage elements induce large area overheads. This section will describe some area-efficient techniques to tackle this challenge. First, we propose an area-efficient architecture for the NVP. Furthermore, we describe two specific implementations referred to as PaCC and SPaC.

11.4.1 Area Challenges and Solutions

The nonvolatile memory cells, such as FeCaps (see Fig. 11.6), magnetic tunnel junctions (MTJ), and floating-gate transistors, usually occupy a quite large area. To remove the performance losts in the normal operations, a hybrid flip-flop is usually adopted, containing a standard flip-flop and nonvolatile cells. It makes the area of NVFFs even bigger. Table 11.2 shows the area overheads of a single NVFF and an NVP under different nonvolatile techniques. It shows that all of them introduce nontrivial area overheads. Note that the area overheads do not consider the variation and reliable issues, which make the chip area even larger. For example, the FeCap has a sandwich structure with a ferroelectric film layer between two metal layers. To reduce error rates during the read and write operations, the ferroelectric film should be large enough. Measurements show that nearly five times area overhead is observed in a commercial FeFF. Recently, Beach et al. [20] pointed out that the MTJ-based memory demonstrates statistical read and write behaviors and requires extra error-correcting units.

Table 11.2 Area challenges under different nonvolatile techniques

Approach	Area of NVFF in DFFs	Area overhead of entire chip (%)
Floating gate [2]	1.4x	19
Magnetic RAM [10]	2x	40
FeRAM [21]	4–5x	90

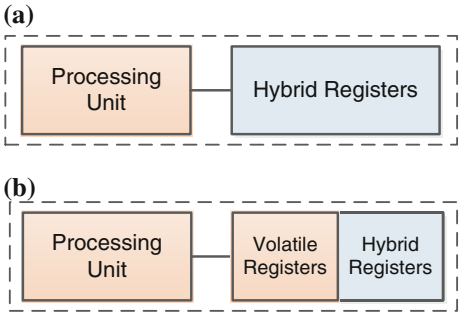


Fig. 11.12 Hybrid architectures for NVP. **a** Hybrid architecture of the fabricated NVP. **b** Partial hybrid architecture for NVP

To quantify the area overheads in an NVP, we assume that an NVFF is α times larger than the original flip-flop and all the NVFFs occupy $\beta(0 < \beta < 1)$ of the total chip area. The area overhead S_{ov} equals to $\beta \times (\alpha - 1)$ when replacing standard flip-flops with NVFFs. In a fabricated NVP [21], β approximates to 20 %, α is near to 5, and $S_{ov} \approx 80\,\%$. Therefore, efficient design techniques are needed to alleviate those area impacts.

Figure 11.12b presents a partial hybrid nonvolatile architecture to deal with the area challenge. In this architecture, we only replace parts of flip-flops with NVFFs so as to reduce the chip area. However, there may be insufficient NVFFs to backup the system states. Data compression and corresponding techniques are proposed to ensure the correct backup and recovery operations. The data compression can be realized in either a software or a hardware approach. In the software approach, the application stores critical data into NVFFs, while keeping other data in the volatile flip-flops. Only critical data are stored when power off. The advantage of the software approach is its flexibility. However, it puts the burden on the application developers to determine which data should be stored. Another solution is to design a specific hardware codec to perform data compression. It can compress data fast and is transparent to the application designers. Thus, we present two concrete designs of compression codec, named PaCC and SPaC, as follows.

11.4.2 An Parallel Compare-and-Compress Codec for Nonvolatile Processors

This section presents an area-efficient architecture referred to as parallel compare-and-compress codec (PaCC) to reduce the number of the nonvolatile registers, and thus the area of the NVP. It consists of an improved RLE algorithm and the corresponding hardware implementation. Furthermore, a state table selection solution is proposed to improve the PaCC's compression ratio.

11.4.3 PaCC Overview

First, we analyze the redundancy in processor states stored in registers, which reveals the potential of register reduction by compression. Let $v_i = 0/1$ represents the value of the ith flip-flop in a processor at a given time. We use $\mathbf{V} = (v_1, v_2, ..., v_n)$ to denote the processor state at that point. Simulations show that over 80 % of the v_i's 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 to 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 $\mathbf{V}^{\text{c}}_{\text{diff}}$ and use fewer NVFFs to store it. We define the compression ratio as

$$CR = \frac{|\mathbf{V}^{\text{c}}_{*}|}{|\mathbf{V}_{*}|}, \tag{11.1}$$

where $\mathbf{V}_{*}$ is any original vector and $\mathbf{V}^{\text{c}}_{*}$ is the compressed version.

The comparison of a conventional NVP and the PaCC architecture is shown in Fig. 11.13. Figure 11.13a shows a straightforward way to implement an NVP which simply augments a volatile processor with FeFFs and a FFC. Such a processor stores the system state $\mathbf{V}$ without compression, and the FeFFs increase area significantly. The PaCC architecture (see Fig. 11.13b) 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 FeFFs. The state table is used to store $\mathbf{V}_{\text{ref}}$, and the compression codec is used to make conversions between $\mathbf{V}_{\text{diff}}$ and $\mathbf{V}^{\text{c}}_{\text{diff}}$, and the comparison is done by the bit-wise XORs. Though the volatile registers, the state table, and the compression codec bring in additional area, the significant reduction in the number of FeFFs leads to a much smaller overall chip area. The operation of a PaCC can be partitioned into the encoding procedure and the decoding one. The encoding procedure accomplishes the following:

- halt the clock to maintain the state vector $\mathbf{V}$ when a power interruption is detected,
- select a reference vector $\mathbf{V}_{\text{ref}}$ from the state table, which can generate the most consecutive zeros after comparison,
- calculate $\mathbf{V}_{\text{diff}}$ by XORing $\mathbf{V}$ and $\mathbf{V}_{\text{ref}}$,

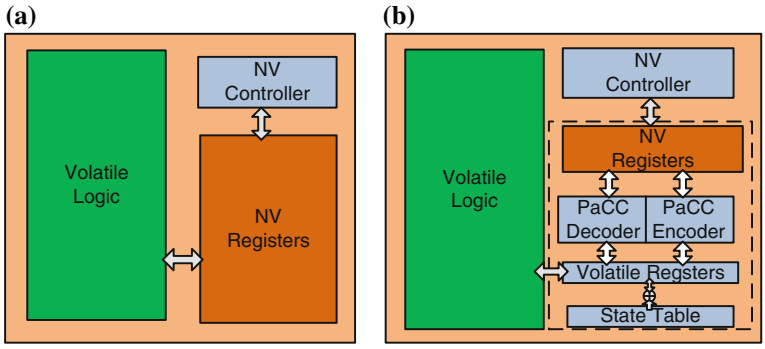


Fig. 11.13 Conventional architecture versus PaCC architecture. **a** Conventional Architecture of NVP. **b** Parallel Compare-and-Compress (PaCC) Architecture of NVP

- compress $\mathbf{V}_{\text{diff}}$ using the PaCC encoder to get $\mathbf{V}^{\text{c}}_{\text{diff}}$, and
- store $\mathbf{V}^{\text{c}}_{\text{diff}}$ into the nonvolatile storages.

The decoding procedure works in the opposite direction. Obviously, the PaCC codec design and the $\mathbf{V}_{\text{ref}}$ selecting are two critical issues in the architecture. We will discuss them as follows.

11.4.4 PaCC Codec

To achieve a fast compression process along with a good CR, we use a threshold-based parallel run-length encoding (PRLE) algorithm to compress the $\mathbf{V}_{\text{diff}}$. Instead of processing the input data stream bit-by-bit, this algorithm introduces a parallel observation mechanism into the traditional RLE to bypass continuous k-bit 0(1) in parallel. In order to further improve the compression ratio, we adopt a threshold value L_{th} to deal with the short 0(1) chains as in [22]. We only encode the 0(1) chains longer than the L_{th} bit because short chains cannot be compressed by PRLE.

Algorithm 10.1 presents the details of PRLE algorithm. The inputs of the algorithm are the differential vector $\mathbf{V}_{\text{diff}}$, the observation window width (OWW) k, and the threshold Lth. The output is the compression result $\mathbf{V}^{\text{c}}_{\text{diff}}$. Variable $\mathbf{S}_{\text{uni}}$ denotes a uniform sequence of all 0's or 1's and $\mathbf{S}_{\text{non}}$ denotes a nonuniform sequence such as {0100101 . . .} which does not contain any 0/1 chains longer than the L_{th} bit.

After the initialization (Line 1), the main body of the algorithm is a "while" loop (Line 2) which checks the end of the input vector. We 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 0/1's. If so, we append all of them to $\mathbf{S}_{\text{uni}}$ to bypass them in one step (Line 4–6). Otherwise we only add the current bit to $\mathbf{S}_{\text{uni}}$ (Line 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 $\mathbf{S}_{\text{uni}}$ (Line 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 re-initialize $\mathbf{S}_{\text{non}}$ and $\mathbf{S}_{\text{uni}}$ (Line 16–20). If reaching the end of $\mathbf{V}_{\text{diff}}$ ($s == n + 1$), we process $\mathbf{S}_{\text{uni}}$ as above (Line 11–13, 16–20). Specially, we call function $Process\_ShortSEQ$ to encode the remaining $\mathbf{S}_{\text{non}}$ (Line 14–15) 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. 11.14. We call them copied segments and encoded segments, respectively.

Algorithm 1: Threshold-based parallel RLE algorithm

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

```
1  Initialization: V^c_diff = φ, S_uni = φ, S_non = φ, s = 1;
2  while s ≤ 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} → S_uni;
6          s = w + 1;
7      else
8          a_s → S_uni;
9          s = s + 1;
10     if (a_s ≠ a_{s+1} && s ≤ n)||(s == n + 1) then
11         if |S_uni| <= L_th then
12             S_uni → S_non;
13             S_uni = φ;
14             if s == n + 1 then
15                 Process_ShortSEQ(S_non) → V^c_diff;
16         else
17             Process_ShortSEQ(S_non) → V^c_diff;
18             Process_LongSEQ(S_uni) → V^c_diff;
19             S_non = φ;
20             S_uni = φ;
```

In this section, we present the hardware design of a PaCC Codec. Figure 11.15 shows the block diagram of our PaCC encoder which is the hardware realization of PaCC algorithm. 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 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

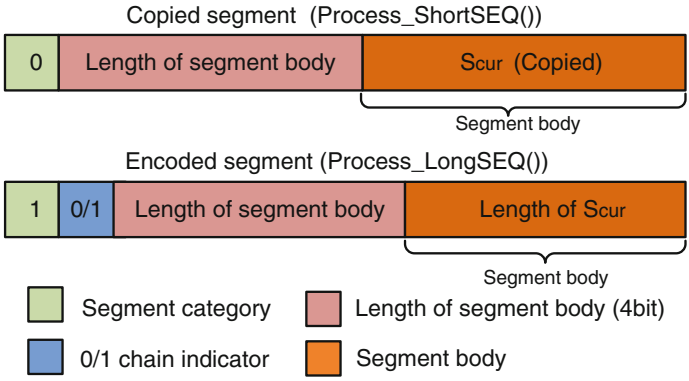


Fig. 11.14 Encoded and copied segment structure. The copied segment is the format used by chains with mixed 0's and 1's of lengths shorter than L_{th}. The encoded segment is used by the chains with lengths longer than L_{th}

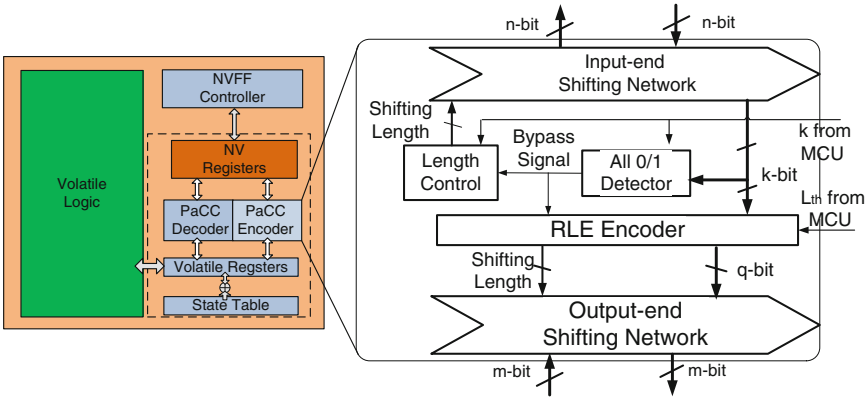


Fig. 11.15 The block diagram of PaCC encoder and its location in the PaCC architecture

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. 11.14). The OWW k and threshold L_{th} are given by the microcontroller (MCU) based on actual applications. Once the RLE encoder accomplishes the compression, it sends the q-bit compressed segment (see Fig. 11.14) 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 part are exchanged and the data flow in the opposite direction. The only difference in the PaCC decoder is that it contains a RLE decoding module instead of the RLE encoding module. Thus, we omit the detail discussion on the decoder part.

11.4.5 State Table Optimization

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 insufficient. 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. 11.16 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 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 so flexible. The other method uses additional nonvolatile 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 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.

Then we will decide how to choose an appropriate reference vector for a specific application. We can formulate the problem as following: assuming β system-state vectors $\{\mathbf{V}_1, \mathbf{V}_2, ..., \mathbf{V}_\beta\}$ should be stored, we need to determine an optimal reference vector $\mathbf{V}_{\text{opt}}$ to minimize the length $L_{\mathbf{V}_{\text{comp}}}$ of the compressed vector under the worst case:

$$\mathbf{V}_{\text{ref, opt}} = \arg\min_{\mathbf{V}_{\text{ref}}} \left(\max_{i=1}^{\delta} L(C(\mathbf{V}_i \oplus \mathbf{V}_{\text{ref}})) \right) \tag{11.2}$$

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 PRLE algorithm.

The direct searching space of $\mathbf{V}_{\text{ref}}$ is very large and function P does not have an analytical pattern, so we develop a naive heuristic to find a solution. Intuitively, the CR of RLE algorithm is related to the number of 0's in $\mathbf{V}_{\text{diff}}$, so we can get a

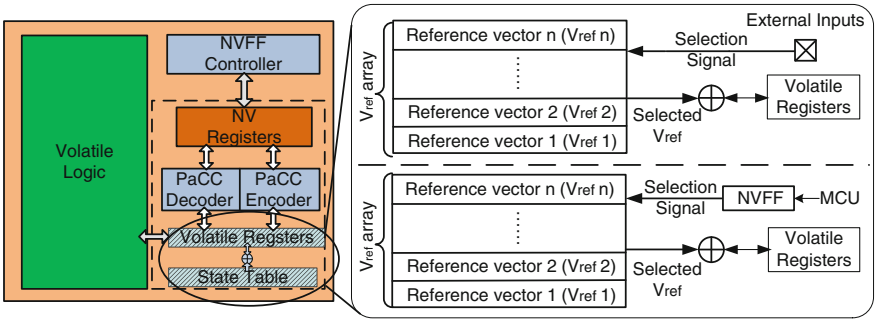


Fig. 11.16 State table structure and the two proposed reference vector selection methods

suboptimal reference vector $\mathbf{V}_{\text{sub,opt}}$ resulting in the most 0's in $\mathbf{V}_{\text{diff}}$ in the global vision. Therefore, we set the ith bit of $\mathbf{V}_{\text{sub,opt}}$ as follows:

$$\mathbf{V}_{\text{sub,opt}}(i) = M(\{j \in 1, 2, ..., \beta | \mathbf{V}_j(i)\}) \tag{11.3}$$

$M(\mathbf{S})$ equals to the majority element in the set $\mathbf{S}$. In our experiments, this method can achieve quite good compression ratio in most cases; however, it may lead to poor results for some special vectors because it ignores the continuity of 0's in $\mathbf{V}_{\text{diff}}$. Some better heuristic algorithms can be explored to address the optimization problem in our future work.

11.4.5.1 Evaluation Results

In this part, we will show the evaluation results of PaCC in chip area and compression speed. We use 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 programs of Fibonacci, sorting and square root from Dalton Project [23]; Rijndael and FFT from MiBench [24], and ZigBee MAC Protocol from Z-Stack [25].

We evaluate the area efficiency of PaCC for the programs. We randomly select 50 state vectors for each program and calculate the optimized reference vector $\mathbf{V}_{\text{ref,opt}}$ based on heuristics in Eq. 11.2. We get the desired number of NVFFs L_{nv} and the area reduction numbers in Table 11.3. Each row represents the results from one program. The columns give out the compression ratio of PRLE, the number of NVFFs, the area reduction ratio of MCU (both MCU only and the whole chip), and the overflow possibility. All data are obtained under the optimal threshold L_{th} for each program. The optimal threshold is the one which results in the smallest number of NVFFs among all the threshold values in [4,50]. In the programs considered, the optimal L_{th} is always 9 or 10. This is due to the fix encoding format in Fig. 11.14.

As shown in Table 11.3, different programs may lead to different numbers of NVFFs (see column 3). Thus, the area savings vary for different programs. By utilizing PaCC, the compression ratio can reach to 19.2 % with the number of NVFFs reduced from 1607 to 308. Based on this reduction, the area saving ratio for the MCU only can be 23.4–30.2 % and the worst case ratio is still above 15 % for the total chip. We conclude that the algorithm is effective to reduce the chip area.

The run-time of encoding and decoding is also important metrics for PaCC. The encoding performance depends on the chosen OWW k. Intuitively, smaller k may not achieve significant reduction in clock cycles while a larger k reduces the opportunity to encounter consecutive zeros or ones. As a result, we can get an optimal k which leads to the smallest number of encoding clock cycles. In our experiments, the optimal k may vary for different programs, and it usually locates in a fixed range of [16–20]. Given the optimal k chosen for each program, Table 11.4 shows the clock cycles of encoding and decoding for different programs. We can see that the encoding

Table 11.3 Evaluation of area efficiency of PaCC architecture

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 11.4 Evaluation of run-time of PaCC codec

Program	Optimal k	Clock cycles				Process time	
		Encode		Decode		On average (μs)	
		Mean	Std	Mean	Std	Encode	Decode
Fibonacci	19	243.2	21.2	97.8	3.3	24.3	9.8
Sorting	16	253.1	25.7	98.1	3.3	25.3	9.8
Sqrt	17	301.8	34.0	101.0	2.9	30.2	10.1
Rijndael	16	211.7	23.4	95.3	3.3	21.1	9.5
FFT	20	190.9	28.9	94.5	3.9	19.1	9.5
ZigBee MAC	20	279.5	42.5	98.5	2.2	27.0	9.9

Assuming the data encoding and decoding procedures runs at 10-MHz clock frequency, the clock cycle statistics is obtained from a circuit simulator
Mean means the average value, *Std* means the standard deviation

process needs extra 200–300 cycles to compress one vector, while the decoding one costs 90–100 cycles. Therefore, the time to store data takes less than 30 μs and the recall takes less than 10 μs at the 10-MHz clock frequency. It maintains the NVP's instant-on/instant-off features.

11.4.6 A Segment-Based Parallel Compression for Backup Acceleration in Nonvolatile Processors

In this section, we will introduce another compression structure referred to as SPaC: a segment-based parallel compression architecture. It achieves trade-offs between the compression time overheads in PaCC and the area overheads in a conventional NVP with full NVFF replacement.

11.4.6.1 SPaC Overview

We give the comparison of different NVP architectures in Fig. 11.17. As Fig. 11.17a shows, the conventional NVP connects each register with a nonvolatile cell. The backup process is totally parallel and fast but leads to nontrivial area overheads due

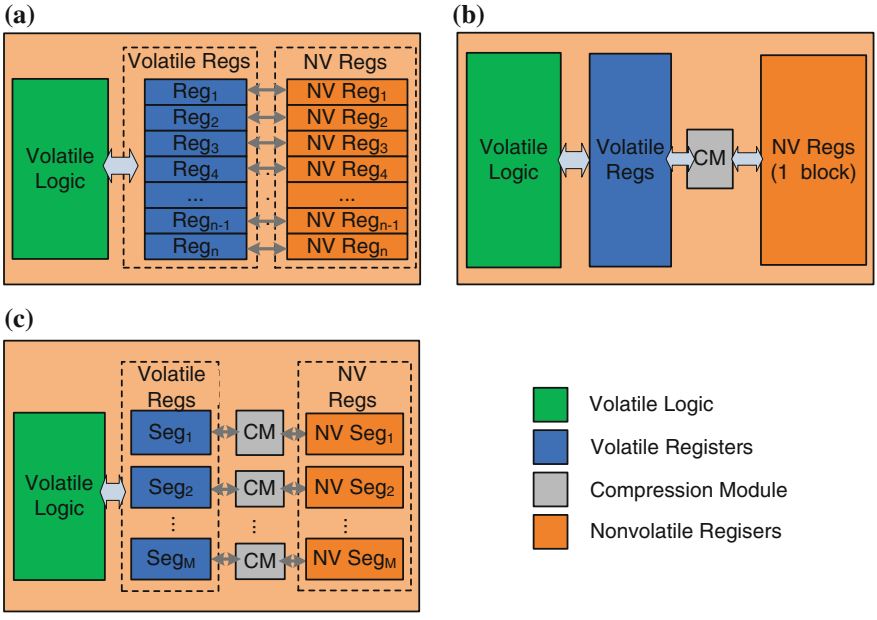


Fig. 11.17 Architecture comparison. **a** Full Replacement Architecture. **b** PaCC Architecture. **c** SPaC Architecture

to a large number of nonvolatile cells. PaCC in Fig. 11.17b uses a compression module (CM) to reduce the number of nonvolatile cells as well as the area. However, the CM compresses the data stream bit-by-bit causing longer backup time. Our proposed SPaC architecture in Fig. 11.17c partitions the registers into several segments and equips each segment with an individual CM. In SPaC, all segments are compressed in parallel to achieve faster backup speed against PaCC. Meanwhile, SPaC reduces the area against the full replacement approach.

Two key metrics of SPaC are the area and backup speed. We evaluate the metrics versus numbers of segments M to show their trends. The results in Figs. 11.18 and 11.19 are based on THU1010N (discussed in Sect. 11.3). Figure 11.18 shows the chip area normalized to the full replacement realization versus M. The area data are approximately linear to the number of segments, because the increasing area primarily comes from the additional CMs. Moreover, the total compression effect of PRLE algorithm degrades when the input vector is divided to more segments, which is another factor inducing the area increase. In our case, the area of SPaC cannot save area when $M > 7$. Figure 11.19 shows the compression speed under different M. Generally speaking, a larger M leads to deeper parallelism and fewer clock cycles. However, when $M > 6$, the speedup by further parallelism is trivial. We use three

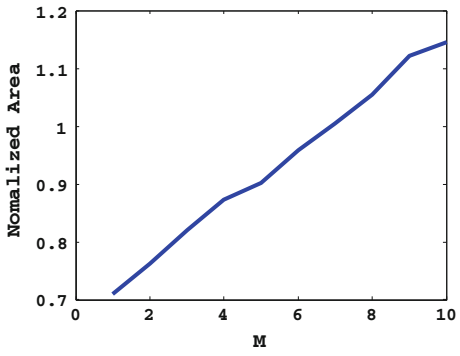


Fig. 11.18 Area evaluation

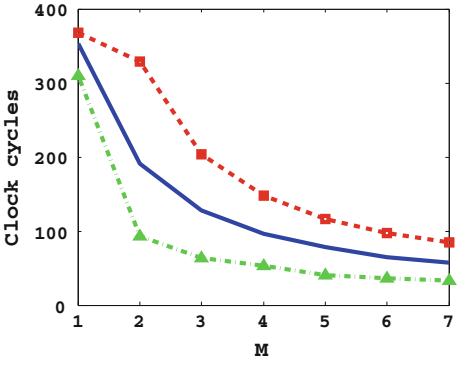


Fig. 11.19 Speed evaluation

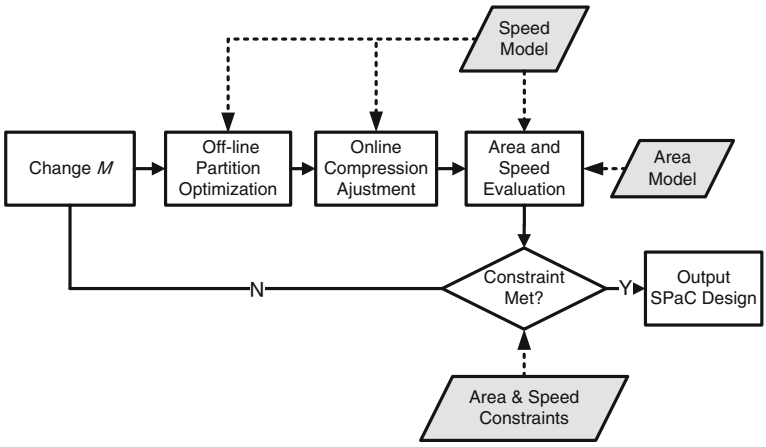


Fig. 11.20 Design flow of SPaC

curves to indicate the average value and the upper/lower bound in Fig. 11.19. The variations come from the input changes at different backup points. If the variation is large, it can significantly degrade the backup speed. This can be solved by an online scheduling controller in the next subsection. Considering both area efficiency and compression speed, the appropriate M is 2 or 3.

Although the data in Figs. 11.18 and 11.19 are based on a specific case, the trends of area and speed are common in other computer architectures (such as MIPS, X86). However, the demarcation point may be different in other processors, because they have different register numbers and architectures, and their area models may be different in other technology processes. Therefore, SPaC can be applied to other processors, but the number of segments should be determined according to the actual design and its requirements. For an actual processor, we propose the design flow for SPaC in Fig. 11.20. As Fig. 11.20 shows, different Ms are evaluated according to the area and speed constraints. We will change the value of M if the constraints are violated. Given a certain M, off-line partition optimization and online compression adjustment are introduced to minimize compression time.

11.4.6.2 SPaC Design

Figure 11.21 shows the detailed diagram of SPaC with M segments. The flip-flops are clustered into M segments denoted as $\{S_1, S_2, ...S_k, B_{k+1}, ..., B_M\}$ and each segment is connected to a CM module for parallel compression. Segments S_i usually have relative small workload variations and do not support compression reallocation. The determination of S_i is based on an off-line algorithm to balance the workloads on each CM. To support dynamic workload adjustment, we design a specific structure to allow the segments to share their CMs if some CMs are idle and others are busy.

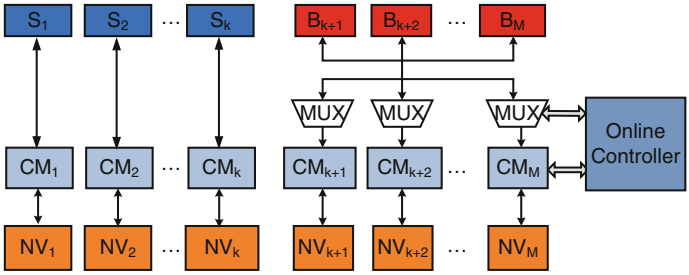


Fig. 11.21 SPaC structure

We denote such shared segments as B_i. To support the CM sharing among segments B_i, we use a set of MUXs to realize the switching operations.

First, we describe the off-line partition algorithm in the following.

Algorithm 2: Off-line Algorithm

Input: $\mathbf{V}$, $\mathbf{M}$, $\mathbf{Var}_{th}$, $\mathbf{loop}_{th}$
Output: $\mathbf{S} = (\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M)$
Variables: $std, time, step, loop$

$\boldsymbol{S}_i = length(\boldsymbol{V})/\boldsymbol{M}$ *for* $i = 1, 2, \ldots, \boldsymbol{M}$; $std = \boldsymbol{S}_{th}$;;
while $std \geq \boldsymbol{S}_{th}$ *and* $loop \leq \boldsymbol{loop}_{th}$ **do**
  $time = CM(\boldsymbol{V}, \boldsymbol{S})$;
  $std = STD(time)$;
  $step = ceil(std)$;
  $\boldsymbol{S}(Indexof(\max(time))) = \boldsymbol{S}(Indexof(\max(time))) - step$;
  $\boldsymbol{S}(Indexof(\min(time))) = \boldsymbol{S}(Indexof(\min(time))) - step$;
  $loop = loop + 1$;

Supposing that we partition the system-state vector V into M segments, S_{th} denotes the threshold of the standard deviation and $loop_{\text{th}}$ denotes the loop limitation. The output vector $\mathbf{S} = (\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_M)$ represents the length of each segment. The variable std denotes the temporary standard deviation under the current partition S; $time = (t_1, t_2, , t_M)$ gives out the average clock cycles of all segments; $step$ is the max step value to change the vector length, and $loop$ is the iterating number. We use the equal partition as the initial S and set std to S_{th}. In each loop, we calculate the compressing time of each segment to get $time$ and its variation std. We find the segment with the maximum average clock cycles in $time$ and reduce its vector length by one step. Similarly, the opposite operation is performed to the segment with the minimum average clock cycles. We keep changing S until std is smaller than S_{th}, otherwise it will return an error message. In case of failures, we either reduce S_{th} or set a larger $loop_{\text{th}}$.

Furthermore, we illustrate the dynamic workload adjustment based on CM sharing. Each shared segment B_i is divided into two parts. One part is shared which is

connected to MUXs. The remaining parts are directly connected to the segments' own CMs. During the compression, the online controller monitors the complete state of each segment. If one segment completes its compression and another is not, the online controller switches the shared part of the slowest segment to the CM of the fastest segment. To avoid area overheads of the multiplexing, the number of shared segments is small. The size of shared parts of each B_i is determined by the compression speed variations.

11.4.6.3 Evaluation Results

We compare metrics of an NVP using equal-size partition, off-line only partition and hybrid off-line/online partition under different Ms in Table 11.5. We can see that the off-line algorithm balances the workloads of different segments effectively while the hybrid algorithm further decreases the variations. As Table 11.5 shows, the off-line-only partition can improve the compression speed by 32 % compared to the equal-size partition. The hybrid strategy further reduces the variations by average 31.7 % and improves the overall speed performance by up to 10 %.

11.5 Nonvolatile Processor Applications

In this section, we describe two typical applications based on an NVP. The first one is a vehicle detection system. The second one is a self-powered sensor node aimed at body area monitoring. These two application systems have unique features differing from traditional sensor nodes, and we list them as follows:

1. Both systems are driven by immediate harvested energy without conventional energy storage devices, such as batteries.
2. Both systems work continuously under frequent power interruptions even using a square-wave power supply.

Those system features are attributed to the features of NVPs. The complexity to design a power system for a NVP-based sensor node can be significantly reduced without AC–DC regulators and energy storages. It implies the potential to reduce the cost and size of the total system.

11.5.1 Vehicle Detection System

The vehicle detection system is based on energy-driven nonvolatile sensor nodes. The whole system is depicted in Fig. 11.22. Each nonvolatile sensor node is equipped with a solar cell energy harvester with no batteries. The energy source can be sunlight outdoors or light sources indoors. Given an energy source, the sensor node continuously

Table 11.5 Compression speed comparison among equal partition, off-line method only and off-line + online method

M	Equally partition			Off-line only				Off-line + Online				
	Avg	3*Std	Sum	Avg	3*Std	Sum	Red. (%)	Avg	3*Std	Sum	Var. red. (%)	Overall red. (%)
2	288.9	31.8	320.7	199.2	22.5	221.7	30.8	203.8	15.6	219.4	30.6	1
3	211.1	24.9	236	130.6	25.4	156	33.8	130.2	17.7	147.9	30.3	5.1
4	155.4	25	180.4	97.2	24.9	122.1	32.3	100.2	17	117.2	31.7	4
5	124.4	28	152.4	80.2	23.1	103.3	32.2	82.3	16.1	98.4	30.3	4.7
6	101.8	18.4	120.2	67.7	22.7	90.4	24.7	67.5	13.6	81.1	40	10.2
7	96.1	24	120.1	58.1	15.9	74	38.3	59.5	11.5	71	27.6	4

Avg average value, *Std* standard deviation, *Var* variation, *Red* reduction

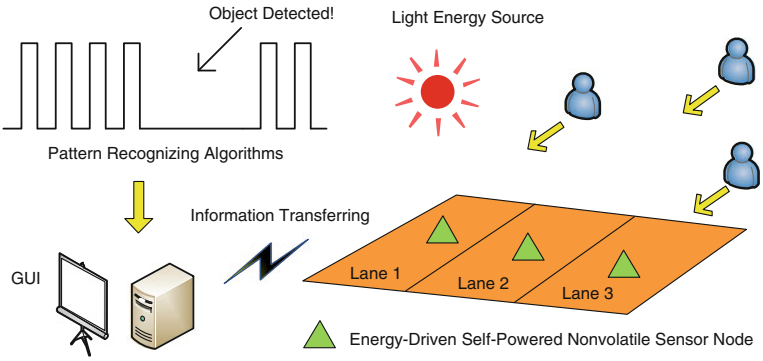


Fig. 11.22 The proposed moving vehicles detection system

counts the number. When a moving vehicle (e.g., a car) comes in between sunlight and sensor node, power to the sensor node is cut off. The nonvolatile sensor node will remember the current state and wait for the moving object to pass by. After that, the power supply is recovered, and the sensor node will continue to count. The counting number and related information can be stored in the local nonvolatile memory or wirelessly transferred to the remote data center. An object recognition algorithm is used in the data center to analyze the object occurring time and other information. A synchronization algorithm should be implemented among the collecting point and sensor nodes. After a certain period of time, the global time should be refreshed and synchronized with each node. A graphic user interface (GUI) is provided to show the real-time detecting results. The most novel technique used in this demo is the NVP-based energy-driven system. This system consists of the energy-harvesting module (solar cell), the power management unit (PMU), and the NVP, shown in Fig. 11.23. At several square centimeters in size, a solar cell is used to provide 6-V and more than 5-mW power supply under medium sunlight. The PMU realized the functions of energy detection and voltage regulation. It measures the energy stored on the capacitor and generates activation signals to the NVP as well as regulates the supply voltage.

To better describe the system working mechanism, we draw the signal timing diagram of the sleep and wake-up actions in Fig. 11.24. In the sleep action, when the PMU detects a power failure, it generates a sleep signal and maintains the power supply via the capacitor until the system state is stored in nonvolatile cells. In the wake-up action, the PMU detects the power recovery and provides power to the NVP until the voltage is stable. After that, it generates a wake-up signal to restart the NVP. According to the measured results, the wake-up action costs less than 100 μs and the sleep action takes around 50 μs, which enables our system to work under a frequently interrupted power supply.

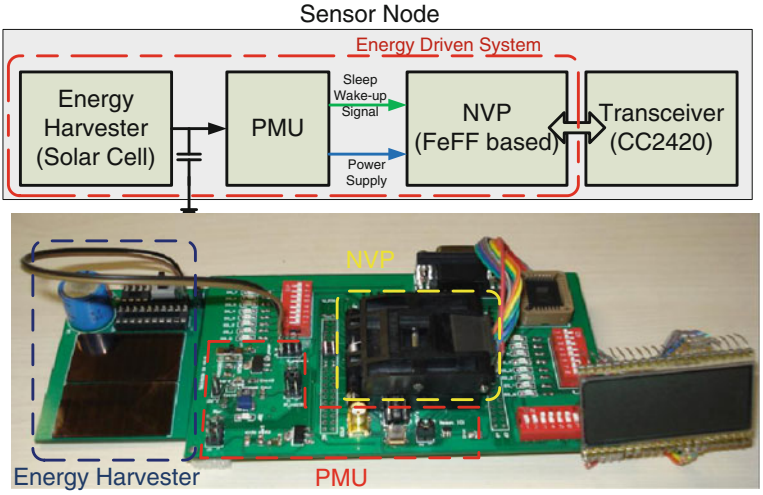


Fig. 11.23 Architecture and realization of energy-driven sensor node

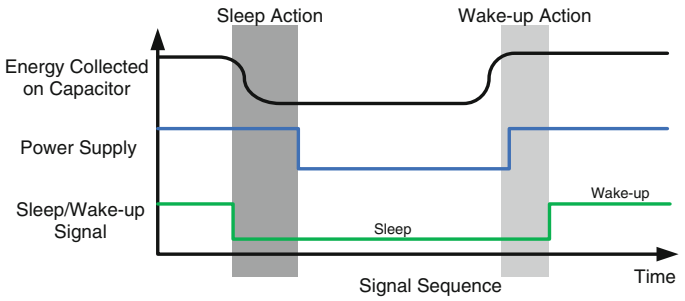


Fig. 11.24 Signal timing chart in sleep and wake-up actions

11.5.2 Self-Powered Body Sensors

Another NVP-based application focuses on the body health monitoring, which is significant to the personal life. Recently, many works have concentrated on the wireless body area network (WBAN) implementation. The body sensor nodes should achieve ultra-low power, low costs, small size, and high reliability. The NVP-based self-powered body sensors can be a promising candidate.

Figure 11.25a shows the block diagram of a self-powered body sensor. Generally, it consists of an energy-harvesting source (EH), a PMU, an NVP, and some peripherals. The NVP provides the node high robustness against power interruptions. The peripherals include several sensors and a RFID. The sensors are used to monitor the medical parameters of a human being and the RFID module enables the node to transmit those data to a sink in a wireless way. The actual sensor node is shown

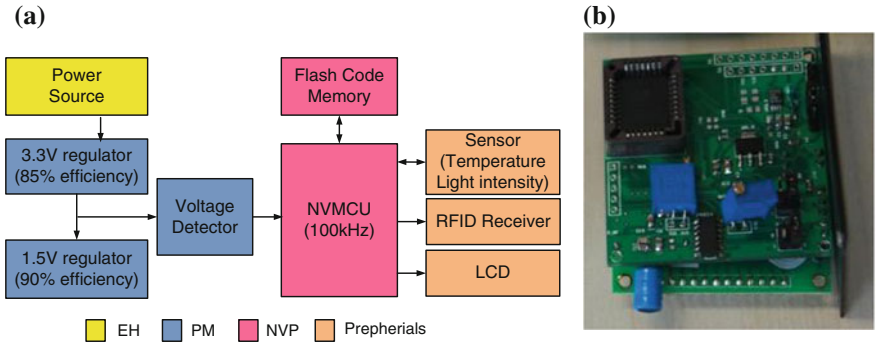


Fig. 11.25 Self-powered body sensor. **a** Block diagram of self-powered body sensor. **b** Actual self-powered body sensor

in Fig. 11.25b. The total size of the node is 50 × 50 × 27 mm. We adopt several power optimizing techniques to enable the sensor node to work under very limited power supply from small-size energy-harvesting devices. The node adopts a DC–DC converter with over 85 % energy-transforming efficiency and a ultra-low-power RFID module. By profiling the power consumption of the sensor node, we find that the Flash code memory contributes a more than 5 mA current under normal operations. As there is a large frequency gap between the NVP and the Flash memory, we design a specific program to power down the Flash memory when it is not read. With this technique, we reduce over 80 % power consumption of the Flash memory and the overall power of the node is reduced to 4 mW. The final demonstration (shown in Fig. 11.26) is a self-powered sensor node with harvested energy from sunlight or vibration. The sensor node monitors the temperature, sunlight duration, and intensity, and it transmits the data into a base station (PC). It mimics the working environment where a human being wears those sensor nodes outdoors or walking. It can collect those medical information reliably without a battery. The users can access those data via a RFID reader in a smart phone or specific devices.

11.6 Related Work

The NVP is a promising approach to realize a nonvolatile-memory-based computing system. Many researchers and companies have evaluated various ways to integrate nonvolatile memories in processors. Flash is a mature high-density nonvolatile memory and is widely used in the mainstream commercial microcontrollers [5, 6]. However, Flash is not suitable to implement distributed NVFFs, because it has drawbacks such as low endurance, slow writing speed, block erasing pattern, and high mask cost. Among existing nonvolatile memories [8], FeRAM and MRAM emerge as the most promising candidates for the NVP. Zwerg et al. [26] embedded a FeRAM into

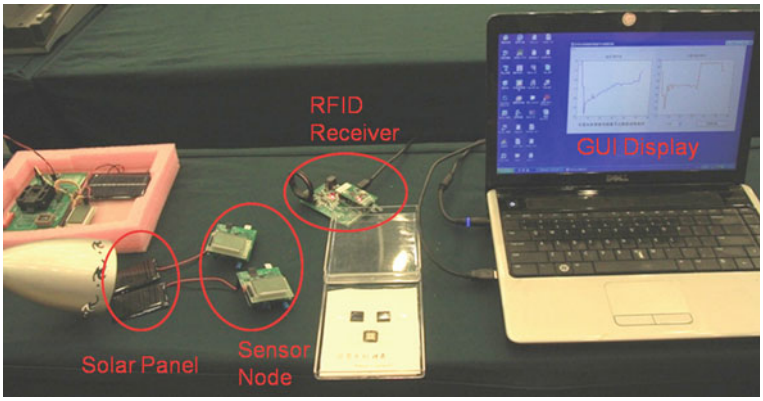


Fig. 11.26 Self-powered body sensor demonstration platform

a microcontroller for better tolerance to power failures. Xu et al. [27] had proposed to adopt STT-MRAM as the last-level on-chip cache in microprocessors. However, the centralized memory architecture cannot provide sufficient bandwidth and fast backup speed in accidental power failures.

In order to achieve faster sleep and wake-up features, some works had concentrated on the register-level nonvolatile memory implementation. Zhao et al. [4] employed MTJ-based flip-flops in FPGAs to achieve rapid start-up. Sakimura et al. [10] developed a magnetic flip-flop (MFF) library for systems-on-a-chip (SoC) design and tested the MFFs in a shifter circuit. Guo et al. [28] conducted an architectural analysis of a STT-MRAM-based processor, including the logic-in-memory, nonvolatile registers, and nonvolatile caches.

Recently, Rohm developed a lifetime-enhanced NVFF by adding a FeCap pair to a standard flip-flop and implemented a nonvolatile counter [1]. The hybrid flip-flop structure does not degrade the performance in the normal operations and prolongs the lifetime of the nonvolatile cells. Wang et al. [12] evaluated an NVP with ferroelectric flip-flops using a compare-and-write policy. Afterward, Yu et al. [2] proposed an evaluation of NVPs based on floating-gate technology. Their analysis demonstrated the performance, area, and power characteristics of an NVP based on the hybrid NVFFs. Simultaneously, Wang et al. [21] fabricated an actual NVP based on the ferroelectric flip-flops and obtained measured results on the sleep and wake-up properties. Most recently, Qazi et al. [29] provided an FIR filter based on ferroelectric flip-flops and demonstrated even faster sleep and wake-up speed.

To further improve the performance of an NVP, some design-optimizing methods are proposed. After observing large area overheads of the hybrid NVFFs, Wang et al. [30] presented a compare-and-compress architecture to reduce the NVP's area and Sheng et al. [31] reported a way to trade off the area overhead and the backup speed in an NVP.

In future, the NVP design may focus on the following aspects: high-speed and reliable NVFF design, hybrid nonvolatile memory architecture, and novel NVP applications.

11.7 Conclusion

In this chapter, we demonstrated the complete design flow to fabricate a ferroelectric NVP. Our experimental results show that the first fabricated NVP can achieve 7 μs sleep time and 3 μs wake-up time with zero standby power, which means over 30–100× speedup on the wake-up/sleep time and 70× energy savings on the backup and recall operations compared with the state-of-the-art industry microcontroller. Meanwhile, the ferroelectric NVP exhibits comparative performance and power consumption in normal operations. Furthermore, we design a PaCC and its variants SPaC to save up to 30 % silicon area in a ferroelectric NVP. Finally, we demonstrate two kinds of battery-less sensor nodes based on the NVP for the first time. They aimed at moving vehicle detection and body sensor applications.

Ferroelectric NVPs can realize energy-efficient computing systems with zero standby power, instant-on features, high resilience to power failures, and fine-grained power management. It has the potential to realize computing system powered by energy-harvesting devices, which eliminates the battery lifetime constraints and becomes a very promising solution for smart sensors and other applications.

Acknowledgments This work was supported in part by the NSFC under grant 60976032 and 61204032, High-Tech Research and Development (863) Program under contract 2013AA013201 and National Science and Technology Major Project under contract 2010ZX03006-003-01.

References

1. Nikkei Electronics Asia: Rohm Develops Non-Volatile Register; Slashes Dissipation. Website: http://techon.nikkeibp.co.jp/article/HONSHI/20080729/155646/.
2. Yu, W., Rajwade, S., Wang, S., Lian, B., Suh G.E., & Kan, E. (2011). A non-volatile microcontroller with integrated floating-gate transistors. In *DSN-W, 2011* (pp. 75–80).
3. Holland, C. First MRAM-based FPGA taped-out. Website: http://www.eetimes.com/General/DisplayPrintViewContent?contentItemId=4200035.
4. Zhao, W., Belhaire, E., Javerliac, V., Chappert, C., & Dieny, B. (2006). A nonvolatile flip-flop in magnetic FPGA chip. In *DTIS, 2006* (pp. 323–326).
5. Texas Instrument: Datasheet of MSP430F522X mixed signal microprocessors (2009).
6. Atmel: Datasheet of AT91SAM9G20-AT91 ARM thumb microcontrollers (2012).
7. Wu, X., Li, J., Zhang, L., Speight, E., Rajamony, R., & Xie, Y. (2010). Design exploration of hybrid caches with disparate memory technologies. *ACM TACO*, *7*(3), 15.
8. ITRS: Roadmap for Nonvolatile Memory. Website: http://www.itrs.net/.
9. Wang, P., Chen, X., Chen, Y., Li, H., Kang, S., Zhu, X. & Wu, W. (2011). A 1.0 V 45 nm nonvolatile magnetic latch design and its robustness analysis. In *CICC, 2011* (pp. 1–4).

10. Sakimura, N., Sugibayashi, T., Nebashi, R., & Kasai, N. (2008). Nonvolatile magnetic flip-flop for standby-power-free SOCs. In *CICC, 2008* (pp. 355–358).
11. Ueda, M., Otsuka, T., Toyoda, K., Morimoto, K., & Morita, K. (2002). A novel non-volatile flip-flop using a ferroelectric capacitor. In *ISAF, 2002* (pp. 155–158).
12. Wang, J., Liu, Y., Yang, H., & Wang, H. (2010). A compare-and-write ferroelectric nonvolatile flip-flop for energy-harvesting applications. In *ICGCS, 2010* (pp. 646–650).
13. Beeby, S. P., Tudor, M. J., & White, N. M. (2006). Energy harvesting vibration sources for microsystems applications. *Measurements of Science and Technology*, *17*(12), R175–R195.
14. Alippi, C., & Galperti, C. (2008). An adaptive system for optimal solar energy harvesting in wireless sensor network nodes. *IEEE Transactions on Circuits and Systems I: Regular Papers, 55*(6), 1742–1750.
15. Lin, K., Yu, J., Hsu, J., Zahedi, S., Lee, D., Friedman, J., Kansal, A., Raghunathan, V., & Srivastava, M. (2005). Heliomote: Enabling long-lived sensor networks through solar energy harvesting. In *ACM SenSys, 2005* (pp. 309–309).
16. Sheikholeslami, A., & Glenn Gulak, P. (1997). A survey of behavioral modeling of ferroelectric capacitors. *IEEE Transactions on Ultrasonics Ferroelectrics and Frequency Control*, *44*, 917–924.
17. Dawber, M., Rabe, K. M., & Scott, J. F. (2005). Physics of thin-film ferroelectric oxides. *Reviews of Modern Physics*, *77*(4), 1083–1130.
18. Du, X. R., & Sheu, B. (2002). Modeling ferroelectric capacitors for memory applications. *IEEE Circuits and Devices Magazine*, *18*(6), 10–16.
19. Texas Instrument: datasheet of MSP430FR573X Mixed Signal Microcontrollers (2011).
20. Beach, R., Min, T., & Horng, C. (2008). A statistical study of magnetic tunnel junctions for high-density spin torque transfer-MRAM (STT-MRAM). In *IEDM, 2008* (pp. 1–4).
21. Wang, Y., Liu, Y., Li, S., & Sheng, X. (2012). A 3us wake-up time nonvolatile processor based on ferroelectric flip-flops. In *ESSCIRC, 2012* (pp. 149–152).
22. Beenker, G., & Immink, K. (1983). A generalized method for encoding and decoding run-length-limited binary sequences (corresp.). *IEEE Transactions on Information Theory, 29*(5), 751–754.
23. T. U. D. project: Benchmark Applications for Synthesizeable VHDL Model. (2006). Website: http://www.cs.ucr.edu/dalton.
24. Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., & Brown, R. (2001). Mibench: A free, commercially representative embedded benchmark suite. In *WWC, 2001* (pp. 3–14).
25. Texas Instrument: Z-stack - Zigbee Protocol Stack. (2009). Website: http://www.ti.com/tool/z-stack.
26. Zwerg, M., Baumann, A., Kuhn, R., Arnold, M., Nerlich, R., Herzog, M., Ledwa, R., Sichert, C., Rzehak, V., Thanigai, P., Eversmann, B.O. (2011). An 82 ua/Mhz microcontroller with embedded FeRAM for energy-harvesting applications. In *ISSCC, 2011* (pp. 334–336).
27. Xu, W., Sun, H., Wang, X., Chen, Y., & Zhang, T. (2011). Design of last-level on-chip cache using spin-torque transfer RAM (STT-RAM). *IEEE Transactions on VLSI System*, 483–493.
28. Guo, X., Ipek, E., & Soyata, T. (2010). Resistive computation: Avoiding the power wall with low-leakage, STT-MRAM based computing. In *ISCA, 2010* (pp. 371–382).
29. Qazi, M., Amerasekera, A., & Chandrakasan, A. P. (2013). A 3.4pJ FeRAM-enabled D flip-flop in 0.13um CMOS for nonvolatile processing in digital systems. *To appear in ISSCC 2013*.
30. Wang, Y., Liu, Y., Liu, Y., Zhang, D., Li, S., Sai, B., Chiang, M., & Yang, H. (2012). A compression-based area-efficient recovery architecture for nonvolatile processors. In *DATE, 2012* (pp. 1519–1524).
31. Sheng, X., Wang, Y., et al. (2013). SPaC: A segment-based parallel compression for backup acceleration in nonvolatile processors. In *DATE 2013* (pp. 865–868).