PAPER

Dynamic Allocation of SPM Based on Time-Slotted Cache Conflict Graph for System Optimization

Jianping WU^{†a)}, *Member*, Ming LING[†], Yang ZHANG[†], Chen MEI[†], and Huan WANG[†], *Nonmembers*

SUMMARY This paper proposes a novel dynamic Scratch-pad Memory allocation strategy to optimize the energy consumption of the memory sub-system. Firstly, the whole program execution process is sliced into several time slots according to the temporal dimension; thereafter, a Time-Slotted Cache Conflict Graph (TSCCG) is introduced to model the behavior of Data Cache (D-Cache) conflicts within each time slot. Then, Integer Nonlinear Programming (INP) is implemented, which can avoid time-consuming linearization process, to select the most profitable data pages. Virtual Memory System (VMS) is adopted to remap those data pages, which will cause severe Cache conflicts within a time slot, to SPM. In order to minimize the swapping overhead of dynamic SPM allocation, a novel SPM controller with a tightly coupled DMA is introduced to issue the swapping operations without CPU's intervention. Last but not the least, this paper discusses the fluctuation of system energy profit based on different MMU page size as well as the Time Slot duration quantitatively. According to our design space exploration, the proposed method can optimize all of the data segments, including global data, heap and stack data in general, and reduce the total energy consumption by 27.28% on average, up to 55.22% with a marginal performance promotion. And comparing to the conventional static CCG (Cache Conflicts Graph), our approach can obtain 24.7% energy profit on average, up to 30.5% with a slight boost in performance.

key words: time-slotted cache conflict graph, SPM controller, energy optimization, dynamic SPM allocation strategy

1. Introduction

Reducing energy consumption remains a major concern of the state-of-the-art SoC (system on chip) design [1], [2]. Many studies have shown that giving consideration to the off-chip memory accesses would contribute to the reduction of the total energy consumption for embedded systems [3]–[5]. In practice, traditional Caches, as one type of the on-chip memories, are implemented to take advantage of a program's locality to reduce accesses to the off-chip memory. However, severe Cache conflicts may happen due to the limitation of Cache's capacity and associative capability, which will in turn degrade the system performance and aggravate the total energy consumption eventually.

Some researchers proposed methods based on the static Cache Conflict Graph (CCG) [6], [7], to copy portions of a program to Scratch-pad Memory (SPM) for performance promotion. However, there are two drawbacks in these methods. Firstly, static CCG can't model the dynamic Cache conflicts' behavior. Most programs have several in-

dependent execution phases and each phase behaves in a different memory accessing pattern. Therefore their methods can't optimize programs according to phases in temporal dimension, which will reduce the utilization efficiency of SPM. Secondly, programs partitioned according to branch instructions may not be suitable for the data sections. The locality of data section is inferior to instructions, which may lead to severe Cache conflicts.

In this paper, a novel Time-Slotted Cache Conflict Graph (TSCCG) [8] is proposed to slice the whole program execution into several Time Slots. And based on the trace files of Cache misses, a conflict graph is established for each phase independently. Then the Virtual Memory System (VMS) is adopted to remap portions of data to SPM according to the TSCCG. When CPU requires data stored in globe, heap or stack segments, the Memory Management Unit (MMU) will decide its physical location in the main memory (MM) or SPM. In order to minimize the swapping overhead of dynamic SPM allocation, a novel SPM controller with a tightly coupled DMA is introduced to issue the swapping operations without CPU's intervention. Finally, Integer Nonlinear Programming (INP) is implemented to choose the most profitable pages, which should be remapped to SPM.

This paper is organized as follows. First, Sect. 1 describes the research background in brief. Section 2 illustrates the difference between our scheme and the previous related works. Then Sect. 3 details the hardware model and the software strategies which are necessary to manage the SPM dynamically. Subsequently, the energy model is established, and INP is implemented to select the optimal solution in Sect. 4. The experimental results will be illustrated and discussed in Sect. 5. Design exploration will be done based on the size change of virtual pages and Time Slots to show what effects they have on the system energy profit in Sect. 6. Finally, Sect. 7 concludes this paper and outlines the future work.

2. Related Works

Cache is transparent to programmers. However, due to its energy hungry, large area and unpredictable execution time, it has been restricted to be used in most embedded systems. By contrast, SPM consumes less energy and areas. In order to take advantage of their complementary features, most high-end embedded microprocessors employ both of SPM and Cache to optimize the system energy consumption and

Manuscript received December 1, 2011.

Manuscript revised April 5, 2012.

[†]The authors are with National ASIC system Engineering Research Center, Southeast University, Nanjing, China.

a) E-mail: herofly2005@163.com

DOI: 10.1587/transinf.E95.D.2039

performance. The optimal algorithms need to manage the SPM in these systems. However, it will change the Cache's behavior when portions of data have been remapped from MM to SPM. Therefore, these methods, which concern SPM only, may not be suitable for those systems with Cache and SPM [9].

Verma et al. presented that the performance and energy overhead could be reduced due to Cache conflicts [6]. They remapped the instructions, which would lead to Cache conflicts to SPM. Based on analysis of the instructions, Memory Objects (MOs) was introduced with its boundary determined by unconditional branch. Each MO was adjusted with the NOP instruction to be alignment with Cache line. Unfortunately, their methods only consider the static situation and could not model the program dynamically, which will in turn degrade the SPM's efficiency. On the other side, their scheme is restricted to instruction partitions and not suitable for data segments due to its partitioning method.

Egger et al. exploited dynamic management methods based on MMU in Seoul National University. Data [4] and instruction [3] were optimized respectively. The MMU manages the system and provides a logical sequential address space which scatters in different physical address spaces. This mechanism could avoid modification of the branch and adding special instructions to the source code during accessing the SPM. Direct-mapped Mini-Cache and SPM were used to replace the conventional 4-way set-associative Cache. Comparing with the latter, the former could obtain considerable system energy profit with a marginal performance promotion. The SPM controller managed the swapping operation of SPM without additional handling instructions to be inserted in user programs. However, the optimal architectures for code and globe data are distinct. Therefore, their methods could not optimize different memory objects as a whole. Moreover, the LDR/STR instructions, used to transport the swapping contents, may contaminate the traditional instruction Cache (I-Cache) and eventually compromise the total profit.

Researchers [8] implemented the TSCCG in SPM dynamic allocation for energy consideration. But it hasn't considered the Time Slot's offset due to performance improvement, and hasn't detailed the composition of energy reduction. It minimized the swapping overhead between SPM and MM assisted by a dedicated DMA channel and the VMS. What effect of the size change for the virtual page and Time Slot has on the system energy profit hasn't been discussed.

Spatial, temporal or hybrid methods have been introduced to bring about the effective utilization of SPM for Priority-Based Preemptive Multi-Task Systems [10]. It relocated the optimal task's functions to SPM, minimized the accesses to external MM. However, it is only for the instruction space which has good spatial and temporal locality, but not suitable for the data partitions, especially for heap and stack data allocated dynamically. What's more, the base optimization granularity is functions which may be too coarse for data partitions dominated by values and array blocks.

Reference [11] minimized inter-task interferences in

SPM utilization for energy reduction of the Multi-Task systems. The SPM space was divided into data and code partitions, each task occupied a separate SPM block. The overlap partitions of different task's block was firstly sliced to active and inactive partitions, and then the overlap partitions was focused on and optimized to minimize the swapping overhead between SPM and MM during task switching. Whereas in their scheme, they need to insert some special API functions in source code to count and save the overlap partitions of context during task switching. This indicates that the source code must be reached and modified firstly, which is infeasible in most cases for software copyright issues. Thus, it couldn't optimize programs non-invasively.

As shown above, previous studies mostly focused on data or instruction partitions but not stack segment. However, experimental results have shown that the most frequently accessed partitions almost concentrated in one or several sequential address spaces of stack. Based on analyzing and tracing the profiling results of the standard test benchmarks, the accesses to stack behave obvious spatial locality. Researchers [12] introduced a method to remap a portion of the stack data to SPM by an arbiter called Shadow Stack Scratch-pad Memory.

Stack and heap data, allocated and used dynamically, cannot be analyzed by the traditional SPM optimization methods. Sjodin et al. [13] partitioned the stack data into 2 parts by the stack pointer, remapped the most frequently accessed ones to SPM. Moreover, the most accessed local variables were treated as globe ones in the function's stack. Soyoung Park [14] proposed a dynamic address-mapping scheme for stack management assisted by the MMU, and without architecture's modification and compiler's assistance. Some researches transformed heap data to stack one [15], [16] and remapped it to SPM, but the efficiency of data type conversion is very low. Dominguez et al. [17] proposed a dynamic allocation strategy to swap some heap data in SPM during the compile time.

Comparing with the earlier researches, Poletti Francesco et al. [5] proposed a method combining hardware with software strategy to manage the SPM dynamically, make full use of the SPM in Multi-thread environment by VMS. DMA was exploited to transport data between SPM and MM to reduce the swapping overhead.

Tiago Rogério Mück et al. [30] allocated the suitable memory level (SPM or external MM) for heap data assisted by the annotations inserted by programmers based on an OS (Operating System). SPM was dynamically allocated during the run-time, and a proportion of energy profit has been obtained on a small cost of performance overhead. However their scheme has two drawbacks. The first one lies in that some special instructions must be inserted to source code manually, which means that the source code must be obtained and modified firstly, this could be a problem in most cases due to the software copyright issues. The second one is that they can only deal with the heap data, or the global data converted to heap ones during system initialization, which will degrade the optimization efficiency and lead

to additional system overhead.

R. Pyka et al. introduced a SPMM (SPM Manager) into an OS for multi-process system [31], to manage the SPM and its allocation optimization. The memory allocation task has been migrated from user applications to the OS. The global data, functions and dynamically allocated data were combined into memory objects. The memory objects were re-allocated according to their optimization efficiency, and the optimal objects were remapped to SPM. A variety of allocation strategies have been studied in their proposed approach. Finally, the Optimal Allocation Strategy based on an ILP (Integer Linear Programming) offline allocation strategy has been developed. With the maximized objective function being constructed and solved, the optimal system energy consumption will be obtained at last.

Cache and SPM, as two dominate on-chip memories, they will complement each other in functions, especially for SPM to Cache, but not confrontational. However, in this paper, they have been considered separately and independently which sounds not reasonable [9], [12], [31], [32]. In experiments, the optimization results of system with Cache or SPM were compared to those systems neither with Cache nor SPM, and this is unfair in most cases. The authors declared that if they extended their scheme for RTEMS OS and standard libraries, the SPM-allocation scheme proposed with their approach would definitely outperform Caches for any kind of applications, this statement is too arbitrary. Actually, a lot of embedded processors and MCUs have adopted this memory architecture in their physical implementation such as IXP network processor of Intel [33], XScale of Marvel and i.MX27 series of Freescale [34] etc.

In this paper, the optimization strategy is based on the VMS, which could analyze the program data as a whole, including heap, stack, globe and constant pool data. The data could be swapped to SPM dynamically without the requirement of compiler's modification and source code. The program will be optimized non-intrusively. The main contribution of this paper lies in two sides. Firstly, dynamic swapping of SPM contents will be more efficient of SPM usage in the limited size. Secondly, our dynamic allocation scheme will be more precise to the time-phase characteristics of programs.

3. Platform

In this section, the hardware model and software strategy will be described in detail. The hardware model consists of an enhanced VMS, a SPM controller with a tightly coupled DMA and a traditional timer. The software strategy sub-section will provide the fundamental basis of this paper, TSCCG. After that, the complete optimization process will be illustrated step by step.

3.1 Hardware Model

As mentioned above, assisted by the traditional VMS, we can remap those memory pages, which may lead to severe

Cache conflicts, to SPM. However, due to the large conventional virtual page size (typical 4 KB/page), the SPM couldn't be managed in fine-grains, and this could eventually degrade the SPM's efficiency. Moreover, the large page size will significantly increase the swapping burden of SPM. The micro-paged MMU (512 B/page) in Ref. [4] is adopted in this paper.

Francesco [5] firstly proposed a dedicated DMA for the data swapping between SPM and MM, which can prevent I-Cache contamination and reduce the performance degradation by increasing the data transportation efficiency. And this method will eventually reduce the data-copy overhead effectively. However, their method had to insert some high level APIs into the source code for DMA's configuration. This required direct access to the source code, which might not be available in most cases for software copyright issues.

In order to minimize the swapping overhead, this paper proposed a novel SPM controller model with a tightly coupled DMA as shown in Fig. 2. The SPM controller model is implemented in C++. As shown in Fig. 1 the system architecture is implemented in C++. Comparing with the ARMulator [18], it could achieve almost 100% accuracy. And the simulation error accuracy could be controlled within $\pm 7\%$ comparing with the physical ARM926E-J chip. While its simulation speed is 3 times of ARMulator, and 8 times of SimpleScalar [19]–[21]. The SPM controller model is constructed by the control logic, a tightly coupled DMA and sets of region registers. The DMA operation will be configured and issued by the SPM controller itself. The control logic is primarily used to load the configuration of current Time Slot from the configure pool; configure the DMA to issue the SPM swapping. The SPM dynamic allocation algorithm would reach the optimal results. And then, C files including configure information will be generated. This information will be added to the program after the program being recompiled and linked. The page swapping will be accomplished by the dedicated DMA. The SPM Management (SPMM) is a section of software code, which implements the results of the dynamic allocation algorithm to the SPM swapping operation during the program execution. It includes three main parts: (1) The page start physical address to be remapped to SPM; (2) The block number of each page to be remapped to SPM; (3) Cycle number of the Timer (Time Slot size).

In this paper, the following three extensions are extended to the traditional SPM controller:

(1) A set of base address mapping registers are designed to record the address of page buffer in MM. After the current page table has been updated, the configure information of program image would be linked by the SPMM to get the actual mapping. DMA's destination will be quickly obtained by reading the write-back address in this register during the write-back process.

(2) A dedicated DMA channel is added to accomplish the data swapping between SPM and MM. Comparing with the traditional LDR/STR instructions. The DMA can minimize the swapping overhead and interrupt latency by

Fig. 1 System architecture with a SPM controller model.

Fig. 2 SPM controller model with tightly coupled DMA.

fully utilizing the burst characteristic of MM and AHB (Advanced High-speed Bus). SPMM reads the configure information to be loaded to MM, loads the update information to DMA controller according to Time Slots, and finishes the DMA's configuration. Thus, it could accomplish data swapping by single or multi-continual DMA operations.

(3) Write dirty-bit was added to control and reduce the swapping overhead. To guarantee the data consistency of SPM and MM, a dirty-bit was used to record the write operation to SPM. When there was a write operation, the dirty-bit of this page will be set to '1'. The SPMM will decide whether to execute DMA operation or not by the dirty-bit in the later Time Slot. On the contrary, bit '0' indicates that the page has not been written, thus the DMA operation should be suspended and overwritten operation to be executed directly.

As shown in Fig. 3, the SPM Region registers includes information of the page size, two flags (valid and dirty bit), the original physical page address to be loaded to the DMA destination address register. The size field which occupies 2-4 bit stands for the page size. The complete process will be presented later.

In addition, this paper introduces a conventional timer to slice the program execution process into several Slots, help us to explicitly model the dynamic characteristics of the Cache conflicts.

Fig. 3 SPM region register.

Fig. 4 Time-slotted cache conflicts graph.

3.2 Software Strategy

The TSCCG introduces a new dimension, TIME into the Cache Conflict Graph (CCG). As shown in Fig. 4, the whole program execution is sliced into several phases; and Conflict Graph is established within each phase independently. In fact, the practical program behaves obvious spatial and temporal locality during the whole execution. Traditional CCG only considers the spatial locality in static within the whole execution process of a program, and can't model the Cache conflicts according to time phases.

To generate TSCCG from the Cache miss Graph, the following steps should be finished:

(1) Slice the program execution into several Time Slots. In theory, the duration of one Time Slot will influence the system optimization. Oversized ones may not reflect the dy-

Fig. 5 Page conflict graph of JPEG-enc in one time slot.

namic execution characteristics of a program. On the other hand, based on our optimization algorithm for SPM swapping, each swapping overhead will be compared with its energy profit, and too small ones will lead to the profit and overhead to be neutralized, thus these small ones may eventually be combined into other big ones.

(2) Count page's conflicts of each Cache line within each Time Slot. And the trace file will be used to determine the page replacement.

(3) Since our optimization scheme manages program in pages, the conflicts of Cache lines should be merged up according to pages. Page is defined as 512 B/page and Cache line as 32 B/line, putting each Cache line's conflicts together to build up the page's conflicts.

(4) Merging Cache lines' conflicts to page ones, getting the conflict graph of each Time Slot. As shown in Fig. 5, it could be found that page 12362 and 12346 have the most conflicts due to the obvious spatial locality of Cache conflicts. There is no need to increase the whole Cache's associative or capacity to remove these conflicts, and only need to remap the frequently conflicted pages to SPM, break up the conflict chains to wipe off these conflicts.

TSCCG $G(T) = (page, E)$ is a directed weighted graph with node set $page = \{page_1, \dots, page_n\}$. T (Time slot) indicates the Time Slot. Each vertex $page_i$ in $G(T)$ corresponds to one page, which is also the granularity of our optimization method. The edge set E contains an edge e_{ij} from node $page_i$ to $page_j$, which denotes that a Cache line belonging to $page_j$ is replaced by the Cache line belonging to $page_i$. The weight m_{ij} of the edge e_{ij} stands for the Cache miss number of $page_i$ that occur due to $page_j$. The weight $dw(ni)$ of

Fig. 6 Interrupt process flow diagram of time slot.

vertex $page_i$ denotes the total access times within $page_j$.

Figure 6 is the swapping operation and control flow within one Time Slot. The three shadow pages will be swapped to SPM in the next Time Slot, and the other pages marked by '0' to be remained in MM. Page A and B have been written dirty in the previous Slot, and they should be written back to their initial physical address recorded in the SPM control registers. Page C has not been written and needn't to be written back. According to the configure information, *PAGE C* should be replaced by *PAGE F* ($0x0030D00$); *PAGE A* replaced by *PAGE D* ($0x0030B700$) and written back to $0x007FFF00$; *PAGE B* swapped with *PAGE E* ($0x0030AA00$), and written back to $0x0030B700$. Completing the swapping operation, the Page Table should be modified to remap the virtual address to the new physical address. Thus the program can access the right ones after the address of *PAGE A-F* has been changed. Exiting the interrupt process, program will access *PAGE A-C* in MM and *PAGE D-F* in SPM. The page's PTE (Page Table Entry) in SPM will be marked as non-Cacheable, which can eliminate the Cache conflicts due to these pages.

4. Energy Model and Algorithm

TSCCG is established to model the Cache conflict behavior of each page according to the profile information. To select the page which causes more Cache conflicts (the most profitable pages), a mathematical model is established from the TSCCG, and some mathematical tools have been used to obtain the optimal solution.

4.1 Energy Model

The system's energy model is firstly established to quantify the fluctuation of system energy when some of the nodes

have been remapped to the SPM. Note that, the energy consumes by those cacheable pages in given Time Slot TS_k could be described as the following equation:

$$\begin{aligned} E(page_i, DCache, TS_k) &= RHit(page_i, TS_k) * E_{DCache_Rhit} \\ &+ WHit(page_i, TS_k) * E_{DCache_Whit} \\ &+ RMiss(page_i, TS_k) * E_{DCache_Rmiss} \\ &+ WMiss(page_i, TS_k) * E_{DCache_Wmiss} \end{aligned} \quad (1)$$

The $R/Whit()$ and $R/WMiss()$ return the number of read/write hit and miss within TS_k . In our scheme, write-through and read allocation strategy are adopted. On this premise, (1) can be simplified as (2):

$$\begin{aligned} E(page_i, DCache, TS_k) &= RHit(page_i, TS_k) * E_{DCache_Rhit} \\ &+ RMiss(page_i, TS_k) * E_{DCache_Rmiss} \end{aligned} \quad (2)$$

Since the read allocation strategy is adopted for Cache, the energy consumption of data read miss should be described as (3):

$$\begin{aligned} E_{DCache_Rmiss} &= 2 * E_{DCache_read} \\ &+ DCache_linesize * (E_{sdram_read} + E_{DCache_write}) \end{aligned} \quad (3)$$

In (3), the second polynomial denotes the energy consumed by filling a new Cache line when read miss happens and the first one stands for the energy consumption of the twice Cache reading operations.

$$\begin{aligned} DMiss(Page_i, TS_k) &= \sum_{page_j \in N(page_i, TS_k)} Miss(page_i, page_j) \\ N(page_i, TS_k) &= \{page_j : \forall nj \in N \& e_{ij} \in E, TS_k \in TS\} \end{aligned} \quad (4)$$

$DMiss(Page_i, TS_k)$ in (4) stands for the Cache miss number of $page_i$ due to $page_j$ in TS_k . Function $N(page_i, TS_k)$ denotes the pages' collection which will lead to $page_i$'s conflicts in TS_k . The total number of CPU access pages in TS_k is a constant described as the weight $dw(n_i)$ of node n_i in (5).

$$dw(n_i) = DHit(page_i, TS_k) + DMiss(page_i, TS_k) \quad (5)$$

Given a page remained in MM, if it will be remapped to SPM, the energy consumption within TS_k should be described as the following equation:

$$\begin{aligned} E(page_i, DSPM, TS_k) &= w(n_i) * E_{spm_read} + l(page_i, TS_k) * [dpagesize \\ &* (E_{sdram_read} + E_{spm_read}) + (1 + W) * E_{dma} \\ &+ W * dpagesize * (E_{sdram_write} + E_{spm_read})] \end{aligned} \quad (6)$$

$$l(page_i, TS_k) = \begin{cases} 1 & \text{if } page_i \text{ is not in SPM yet} \\ 0 & \text{otherwise} \end{cases}$$

In (6), $l(page_i, TS_k)$ stands for the page swapping overhead. According to a special page, this overhead doesn't occur in each Time Slot, such as if $page_i$ has been swapped to SPM in the previous Time Slot, thus it needn't to be transported any more. The factor W ($0 \leq W \leq 1$), obtained by experiments, is the probability of writing back a single page.

4.2 Mathematical Model

In order to select the most profitable pages, the two energy models (Cache and SPM) are combined into a unified one. The page's location in memory hierarchy is described by $p(page_i, TS_k)$.

$$p(page_i, TS_k) = \begin{cases} 1 & \text{if } page_i \text{ is in Cache} \\ 0 & \text{if } page_i \text{ is in SPM} \end{cases} \quad (7)$$

When $page_i$ is accessed in TS_k , the energy consumption should be described as (8).

$$\begin{aligned} E(page_i, TS_k) &= p(page_i, TS_k) * E(page_i, DCache, TS_k) \\ &+ [1 - p(page_i, TS_k)] * E(page_i, DSPM, TS_k) \end{aligned} \quad (8)$$

$E(page_i, DCache, TS_k)$ and $E(page_i, DSPM, TS_k)$ in (8) have been illustrated in (2) and (6) respectively.

$E(TS_k)$ denotes the total energy consumption within TS_k :

$$E(TS_k) = \sum_{page_i \in N(page_i, TS_k)} E(page_i, TS_k) \quad (9)$$

Only $p(page_i, TS_k)$ and $l(page_i, TS_k)$ in (9) are unknown parameters. For the SPM size is limited, not all of the page nodes in TSCCG will be guaranteed into SPM. The constraint of SPM size is described as following:

$$\sum_{i \in N} pagesize * [1 - p(page_i, TS_k)] \leq spmsize \quad (10)$$

Now the mathematical model can be described as (11). The goal of the dynamic allocating scheme is to minimize the value of (9) with the constraint condition as (10).

$$\begin{cases} \min \left\{ E(TS_k) = \sum_{page_i \in N(page_i, TS_k)} E(page_i, TS_k) \right\} \\ s.t. \sum_{page_i \in N(page_i, TS_k)} pagesize * [1 - p(page_i, TS_k)] \\ \leq spmsize \\ p(page_i, TS_k) = 0, 1 \quad i \in N \end{cases} \quad (11)$$

4.3 Integer Nonlinear Programming

It is shown that the solution of the energy model is a typical optimal issue in mathematical model (11). Since there is a quadratic term of (12) in (9), the mathematical problem in

(11) becomes a typical integer nonlinear problem.

$$\begin{aligned}
 & p(page_i, TS_k) * \sum_{page_j \in N(page_i, TS_k)} Miss(page_i, page_j) \\
 &= p(page_i, TS_k) * \sum_{\substack{page_j \in N(page_i, TS_k) \\ page_j \neq page_i}} p(page_j, TS_k) \\
 & \quad * dw(n_j)] \\
 &= \sum_{\substack{page_j \in N(page_i, TS_k) \\ page_j \neq page_i}} p(page_i, TS_k) * p(page_j, TS_k) \\
 & \quad * dw(n_j)] \quad (12)
 \end{aligned}$$

There is a new variable $P(page_i, page_j, TS_k)$ introduced to fix the nonlinear issue by transforming the nonlinear problem to a linear one in research [2]. However, they pulled in more nodes and constraints during the transformation. Given there is N nodes in a previous slot, there will be $N * (N - 1)$ nodes of $P(page_i, page_j, TS_k)$ to be inserted in the current slot. The total variable number will be as (13):

$$N + N * (N - 1) = N * N = N^2 \quad (13)$$

Since each variable $P(page_i, page_j, TS_k)$ corresponds to three constraints, $N * (N - 1)$ variables of $P(page_i, page_j, TS_k)$ will bring $N * (N - 1) * 3$ ones, thus the total constraints will be:

$$1 + N * (N - 1) * 3 \quad (14)$$

When there are considerable nodes in the conflict graph, the problem would be very difficult to be solved due to too many variables and constraints. In this paper, MATLAB Toolbox *bnb20* [22] is used to tackle the nonlinear problems in (11), thus the optimal pages which minimize the objective function to be obtained.

4.4 Time Slot Adjustment

After remapping profitable pages into SPM, most of the Data Cache conflicts will be eliminated. The total energy consumption of memory subsystem decreased without any performance degradation. This will compress the execution time of each slot. Therefore, the start time of each slot should be revised to align the current one to the previous one before optimization. Before and after optimization, the gap of the start time would be figured out by the iterative solution, and regulated with the SPM controller. The iteration process is shown in Fig. 7.

5. Experimental Results

Since our optimization scheme focuses on the system energy reduction by dynamically remapping pages which may cause severe Cache conflicts to SPM according to the dynamic program execution characteristics. And the optimization results may be clearly illustrated by the comparison of

Fig. 7 Iteration process.

Fig. 8 The comparison of cache Misses' distribution for combine.

Fig. 9 The Energy distribution of combine.

Cache misses' number and distribution, before and after optimization.

In real applications, there may be more than one program to be executed and dynamically switched from each other during the run-time. For this reason, we select benchmark of JPEG-dec, MPEG2, FFT and CRC32 to construct a new benchmark called Combine. In Combine, the four programs will be executed in sequence, and their multi-phase characteristics can be effectively utilized by the program dynamic management. As shown in Fig. 8 (a) and Fig. 8 (b), the Cache misses' distribution is compared with each other in the time of before and after optimization. X axis stands for Time Slots, Y axis for Data Page address, and Z axis is for Cache misses' number.

Figure 9 (a) and 9 (b) describe the system energy distribution, before and after optimization. The total energy has been reduced by 41.41%. 8KB-1A-8K stands for 8KB-1A

(a) 3D Map of D-Cache Misses of 8KB-1A Cache

(b) 3D Map of D-Cache Misses of 4KB-1A-4K

Fig. 10 D-Cache Misses' distribution Comparison in 8KB-1A Cache and 4KB-1A-4K for Benchmark Dijkstra.

Cache coupled with 8 KB SPM. The accesses to external MM have been mostly reduced, and the energy consumed by accessing the MM and off-chip bus has been significantly reduced [23], [24], and the energy consumption of the CPU Core has been reduced due to the reduction of execution time. Since these pages accessed through Cache, which may cause severe Cache conflicts, have been remapped to SPM, the energy of D-Cache has been decreased in a large proportion. On the other side, comparing with Cache, there will be less energy consumed by SPM access. Therefore, the total energy consumption of data portions would be reduced further. Some instruction pages would be remapped to SPM and marked as non-Cacheable, and accessed from SPM directly, thus the energy consumption of I-Cache will be reduced as an additional profit.

As shown in Fig. 10, Fig. 10(a) is the Cache misses' mapping for the benchmark of Dijkstra before optimization, and Fig. 10(b) stands for the Cache misses' distribution after optimization. Comparing Fig. 10(a) with Fig. 10(b), it is shown that there is obviously Cache conflicts in instruction and stack space before optimization. After configured with another 4 KB SPM, implemented our dynamic optimization algorithm, the conflicts in instruction space have been almost eliminated, and the conflicts in stack space being reduced over 75%. For heap space, the conflicts have been reduced over 50% due to most conflict pages being remapped to SPM.

We take 14 benchmarks from MediaBench [25] and MiBench [26] to analyze our optimization results quantitatively. D-Cache is configured as 4KB-1A-4K in experiments. The optimization results are compared with 8KB-1A, 8KB-2A and 8KB-4A set-associative D-Cache. Though our method focuses on the energy optimization, the system performance has also been promoted due to the reduction of execution time. Although energy consumption of D-Cache occupies a small proportion in the total energy, the accesses to external MM caused by these Cache conflicts will contribute a considerable proportion to the total system energy.

The processor's energy is based on Ref. [27]. The en-

ergy consumed by Cache, TLB and SPM is obtained from CACTI 6.5 [28], and the main parameters of CACTI are shown in Table 4. The energy consumption of MM and system off-chip bus could be found in literature [23] and [24]. We ignore the energy consumed by SPM controller. According to literature [29], only the memory access energy, issued by DMA, will be included. After optimization, the energy consumption of the MM and off-chip bus has been reduced significantly with the additional SPM accessing energy neglected.

The energy profits of our optimization method are shown in Table 1 and Fig. 11. Comparing 4KB-1A-4K with 8KB set-associative Caches, the total energy consumption has been reduced by 27.28% on average, up to 55.22% on maximum. As shown in Table 2 and Fig. 12, the system performance has been improved by 9.8% on average after implementing our optimization method.

For the difference in optimization objects, simulation platforms, algorithms etc. proposed in different approaches, it is difficult to simply compare the experimental results of different approaches with each other directly. Our TSCCG counts up the Cache conflicts in temporal dimension based on the static CCG (Cache Conflict Graph) proposed by Verma et al. [6]. As mentioned in Sect. 6.2, the duration of a time slot will influence the optimization results proposed with our TSCCG. Too short will lead to data pages selected and placed on SPM not changed at most time. On the other side, oversized one will make the dynamic algorithm degenerate into a static one. Therefore, we can easily compare our TSCCG with the static CCG. The experimental comparison between our TSCCG and static CCG for the benchmark of Combine is shown in Table 3.

The improvement from static CCG to TSCCG is impressive as shown in Table 3. For the six configurations, our TSCCG could reduce the system energy by 24.7% on average, up to 30.5% on maximum; the system performance has been improved by 5.5% on average, up to 6.8% compared to the static CCG. We can come to the conclusion that our TSCCG is more suitable for program reality than the tradi-

Table 1 Energy comparison of benchmarks in 4KB-1A-4K and 8 KB set-associative D-cache configurations (Unit: nJ).

	Adpcm	Blowfish	JPEG-enc	CRC32	Jpeg-dec	FFT	MP3	MPEG2
4K1A+4K	5.31E+10	4.09E+10	4.28E+10	5.76E+10	9.55E+10	9.27E+10	1.53E+11	3.35E+10
8K1A	5.56E+10	7.01E+10	5.09E+10	1.29E+11	1.23E+11	1.24E+11	2.26E+11	5.64E+10
8K2A	5.30E+10	6.62E+10	4.75E+10	1.28E+11	1.14E+11	1.22E+11	2.23E+11	5.29E+10
8K4A	5.19E+10	6.43E+10	4.61E+10	1.26E+11	1.06E+11	1.21E+11	2.21E+11	5.19E+10
	MPEG4	Rijndael	SHA	Susan	H263-dec	Combine		
4K1A+4K	7.84E+10	3.28E+10	2.66E+10	9.04E+10	2.10E+11	5.76E+10		
8K1A	1.02E+11	4.59E+10	3.32E+10	1.04E+11	2.59E+11	9.91E+10		
8K2A	9.85E+10	3.33E+10	3.25E+10	9.89E+10	2.51E+11	9.50E+10		
8K4A	9.71E+10	3.28E+10	3.22E+10	9.76E+10	2.48E+11	9.34E+10

Fig. 11 Energy comparison in 4KB-1A-4K and 8 KB set-associative caches.**Table 2** Performance comparison of benchmarks in 4KB-1A-4K and 8 KB set-associative D-cache configurations (Unit: Cycle).

	Adpcm	Blowfish	JPEG-enc	CRC32	JPEG-dec	FFT	MP3	MPEG2
4K1A+4K	4.76E+07	4.20E+07	3.21E+07	5.62E+07	7.50E+07	5.76E+07	1.01E+08	2.55E+07
8K1A	5.51E+07	4.81E+07	3.44E+07	7.01E+07	7.86E+07	6.30E+07	1.15E+08	3.01E+07
8K2A	5.24E+07	4.50E+07	3.30E+07	6.86E+07	7.47E+07	6.15E+07	1.13E+08	2.74E+07
8K4A	5.08E+07	4.31E+07	3.21E+07	6.66E+07	7.14E+07	6.02E+07	1.12E+08	2.65E+07
	MPEG4	Rijndael	Sha	Susan	H263-dec	Combine		
4K1A+4K	6.38E+07	2.78E+07	2.56E+07	7.78E+07	1.24E+08	2.88E+07		
8K1A	6.87E+07	3.29E+07	2.58E+07	8.36E+07	1.27E+08	3.26E+07		
8K2A	6.54E+07	2.43E+07	2.51E+07	7.92E+07	1.21E+08	3.16E+07		
8K4A	6.33E+07	2.36E+07	2.46E+07	7.69E+07	1.17E+08	3.07E+07		

tional static CCG.

In theory, static approach could obtain the optimal system energy by remapping all the Cache conflict pages to SPM if there is adequate SPM space to use. While our design philosophy lies in that we not only consider the system energy but also the limited chip area and cost in our scheme, fully utilize the temporal locality of Cache conflicts, slice the whole program execution into a series of Time Slots,

dynamically swap the optimal pages to SPM and replace them out when they are not profitable any more, make full use of the limited SPM, improve its efficiency, taking the constraints of die area and cost etc. into account when optimize the system energy, and finally achieve the highest cost-effective of on-chip resources.

Fig. 12 Performance comparison in 4KB-1A-4K and 8 KB set-associative caches.

Table 3 Experimental comparison of TSCCG and static CCG for combine.

Configuration	Static CCG		TSCCG		Optimization Results	
	performance (cycles)	energy (nJ)	performance (cycles)	energy (nJ)	Performance Gains (%)	Energy Gains (%)
4KB-1A-2KSPM	3.01E+07	7.73 E+10	2.88 E+07	6.10 E+10	4.33%	21.02%
4KB-1A-4KSPM	3.01 E+07	7.74 E+10	2.87 E+07	5.76 E+10	4.65%	25.54%
4KB-1A-8KSPM	3.01 E+07	7.01 E+10	2.88 E+07	5.66 E+10	4.32%	19.36%
8KB-1A-2KSPM	3.00 E+07	7.39 E+10	2.81 E+07	5.55 E+10	6.33%	24.87%
8KB-1A-4KSPM	3.00 E+07	7.39 E+10	2.80 E+07	5.14 E+10	6.76%	30.45%
8KB-1A-8KSPM	3.00 E+07	6.79 E+10	2.80 E+07	4.98 E+10	6.81%	26.69%
Average					5.53%	24.66%

6. Design Exploration

6.1 Page Size Select

The minimal control granularity of SPM is decided by the virtual page size, which will make the on-chip memory to be efficiently utilized by the VMS. To page-managed system, the page size is critical in TLB (Translation Look-aside Buffer) design. For smaller page, the address space attached to each PTE will be reduced accordingly. If TLB is not redesigned under this situation, accesses to MM will increase dramatically due to the frequently TLB misses, which will aggravate the burden of system energy. On the other side, the address space of each PTE will expand according to larger page size, while the swapping overhead of SPM will be more expensive as the following two negative effects:

(1) Increase the actual page swapping overhead. The cost of loading pages to SPM will dramatically increase for large pages.

(2) Decrease the efficiency of SPM. In this paper, SPM is used to load the pages which will lead to severe Cache conflicts. While these conflicts just lies in one or several Cache lines, if the page size is too large, the proportion of SPM's effective unit will be reduced, thus the optimal result may not be reached.

To get the appropriate TLB hierarchy, associative and entries under fixed page size, we did some design space exploration based on 7 Benchmarks. The benchmark of Dijkstra, as an example shown in Fig. 15, illustrates that, whether for the L1 (Level 1) or L2 TLB, configured with larger capacity and higher associative, will lead to less overhead for small page management. Since the TLBs are essentially full/set associative Cache, larger capacity and higher associative will cause larger area, more accessing energy and latency. In practice, we select 16 entries for D-uTLB, 8 entries for I-uTLB and 64 entries, 32 set-associative L2 unified-TLB, 512B for page. Comparing with TLB for 4 KB page, the TLB energy consumption for 1 KB page is 1.32% higher on average, 3.79% and 8.80% for 512B and 256B page respectively as shown in Fig. 16.

As shown in Fig. 13, "Non-optimization_256" denotes the system energy in 8KB-1A with 256B page before optimization; "Optimization_256" stands for the system energy in 4KB-1A-4K with 256B page after optimization. "Non-optimization_512" and "Optimization_512" stand for the system energy in 8KB-1A and 4KB-1A-4K with 512B page, before and after optimization respectively. Comparing "Non-optimization_256" with "Optimization_256", and "Non-optimization_512" with "Optimization_512", it is shown that the system energy of 512B page is lower than 256B one whenever before or after optimization. After op-

Fig. 13 Energy comparison for systems with different MMU page size.

Fig. 14 Energy comparison of systems with different time slot in 4KB-1A-4K configuration.

Table 4 Parameters of CACTI6.5 for 4KB-1A cache.

Parameter Name	Configure Value
Cache size	4096B
Block size	32B
Associative	1 way
Read only ports	0
Write only ports	0
Read write ports	1
Cache banks (UCA)	1
Technology	90nm
Temperature	350K
Tag size	42bit

Fig. 15 System energy map of dijkstra in different TLB configuration.

timization, the energy profit is 24.52% and 27.28% on average for system with 256B and 512B page respectively. Considering the TLB energy consumption and system energy

profit in different page size configuration, trading off these two sides, 512B page is taken in this paper.

Fig. 16 TLB energy comparison for different page size.

Fig. 17 Cache Misses' distribution in different time slot (Unit: cycle).

6.2 Time Slot Size Discussion

The duration of a Time Slot will influence the optimization results of TSCCG. Too short will lead the data pages which have been selected and placed on SPM, to not change at most of the time. On the other side, oversized one will make the dynamic algorithm degenerate into a static one. As shown in Fig. 17, the benchmark of JPEG-enc is selected to illustrate the temporal and spatial locality of D-Cache miss patterns. When the memory access pattern is simulated with the most appropriate period, it appears obvious periodic characteristics which could be used by SPM to allocate the optimal partitions. As described in Sect. 3.2, based on our optimization algorithm, we did some design exploration for the time slot size. The Cache misses' distribution in Time Slot of 2,500,000, 2,000,000 and 1,000,000 cycles is compared with each other. The horizontal axis stands for time and the vertical axis for address space. Every small box denotes the Cache misses' number within this page in current Time Slot. The grey degree of each small box stands for the Cache conflict's severity. When Time Slot is set to 2,500,000 cycles, the conflicts keep high severity within some pages; but for 1,000,000 cycles, the conflicts show ap-

parent periodic characteristic.

As shown in Fig. 14, the system energy is normalized by dividing the energy of 2,500,000 cycles and compared with each other. To most benchmarks, the system energy has the optimal result in time slot of 1,000,000 cycles. But when the Slot size is reduced further, the swapping overhead and energy profit will be neutralized, and small slots may eventually be combined into big ones. In this paper, based on a large number of experiments, 1,000,000 cycles is selected as the Time Slot in our experiments.

7. Conclusions and Future Work

In this paper, we introduce a VMS to remap those data pages, which can cause severe Cache conflicts within a Time Slot, to the SPM according to our innovated TSCCG. Then, a novel SPM controller Model with a tightly coupled DMA is implemented to minimize the swapping overhead of dynamic SPM allocation. Afterwards, we introduce the INP to select the most profitable pages. Finally, based on analyzing the Cache and SPM Co-architecture system, we explore how to select the appropriate virtual page size and the duration of a Time Slot to obtain the optimal system energy. This work might provide reference for the optimization of CPU design. The experimental results show that our approach can reduce a great deal of energy with a slight boost in performance. For the future work, according to the optimization algorithm and program execution characteristics, the size of virtual page and Time Slot should be dynamically adjusted to simulate the real program more realistic. Therefore, the system energy and performance can be further improved.

Acknowledgments

This work is sponsored by Qing Lan Project.

References

- [1] S. Segars, "Low power design techniques for microprocessors," International Solid-State Circuits Conference Tutorial, pp.1-67, Feb. 2001.
- [2] ARM Ltd. ARM Processor Core Overview [DB/OL]. <http://www.arm.com/products/CPUs/>
- [3] B. Egger, C. Kim, and C. Jang, "A dynamic code placement technique for scratchpad memory using postpass optimization," Proc. 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems, pp.223-233, New York, NY, USA, Oct. 2006.
- [4] H. Cho, B. Egger, J. Lee, and H. Shin, "Dynamic data scratchpad memory management for a memory subsystem with an MMU," Proc. 2007 ACM Conference on Languages, Compilers, and Tools for Embedded Systems, pp.195-206, New York, NY, USA, July 2007.
- [5] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and J.M. Mendias, "An integrated hardware/Software approach For run-time scratchpad management," Proc. Design Automation Conference, pp.238-243, San Diego, CA, USA, June 2004.
- [6] M. Verma, L. Wehmeyer, and P. Marwedel, "Cache-aware scratchpad allocation algorithm," Proc. Conference on Design, Automation and Test in Europe, vol.2, pp.1264-1269, Washington, DC, USA, March 2004.

- [7] M. Verma, L. Wehmeyer, and P. Marwedel, "Cache-aware scratchpad-allocation algorithms for energy-constrained embedded systems," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol.25, no.10, pp.2035–2051, Oct. 2006.
- [8] H. Wang, Y. Zhang, C. Mei, and M. Ling, "Energy-oriented dynamic SPM allocation based on time-slotted cache conflict graph," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Leuven, Belgium, pp.598–601, March 2010.
- [9] Y. Ishitobi, T. Ishihara, and H. Yasuura, "Code placement for reducing the energy consumption of embedded processors with scratchpad and cache memories," *IEEE/ACM/IFIP Workshop on Embedded Systems for Real-Time Multimedia, ESTIMedia*, pp.13–18, Salzburg, Austria, Oct. 2007.
- [10] H. Takase, H. Tomiyama, and H. Takada, "Partitioning and allocation of scratch-pad memory for priority-based preemptive multi-task systems," *Proc. Design Automation and Test in Europe (DATE)*, Leuven, Belgium, pp.1124–1129, March 2010.
- [11] L. Gauthier, T. Ishihara, and H. Takase, "Minimizing inter-task interferences in scratch-pad memory usage for reducing the energy consumption of multi-task systems," *Proc. International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*, pp.157–166, New York, NY, USA, Oct. 2010.
- [12] M. Ling, X. Shi, and Y. Zhang, "Shadow stack scratch-pad-memory for low power SoC," *Fifth IEEE International Symposium on Embedded Computing*, pp.394–401, Beijing, China, Oct. 2008.
- [13] J. Sjödin and C.V. Platen, "Storage allocation for embedded processors," *Compiler and Architecture Support for Embedded Computing Systems*, pp.15–23, New York, NY, USA, Nov. 2001.
- [14] S. Park, H.-woo Park, and S. Ha, "A novel technique to use scratchpad memory for stack management," *Proc. Conference on Design, automation and test, EDA Consortium*, pp.1478–1483, San Jose, CA, USA, April 2007.
- [15] J.-D. Choi, M. Gupta, M. Serrano, V.C. Sreedhar, and S. Midkiff, "Escape analysis for java," *Proc. 14th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pp.1–19, New York, NY, USA, Oct. 1999.
- [16] N. Hallenberg, M. Elsmann, and M. Tofte, "Combining region inference and garbage collection," *Proc. ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pp.141–152, New York, NY, USA, June 2002.
- [17] A. Dominguez, S. Udayakumaran, and R. Barua, "Heap data allocation to scratch-pad memory in embedded systems," *J. Embedded Computing (JEC)*, vol.1, no.4, pp.521–540, May 2006.
- [18] ARMulator [CP/OL]. <http://www.arm.com/support/ARMulator.html>.
- [19] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: an infrastructure for computer system modeling," *Computer*, vol.35, no.2, pp.59–67, Feb. 2002.
- [20] F.R. Boyer, L.P. Yang, E.M. Aboulhamid, L. Charest, and G. Nicolescu, "Multiple SimpleScalar processors, with introspection, under SystemC," *2003 IEEE 46th Midwest Symposium on Circuits and Systems*, pp.1400–1404, Cairo, Egypt, Dec. 2003.
- [21] N. Abbas, S. Tambe, and J.E. Cook, "Using DDL to understand and modify SimpleScalar," *Proc. 11th Working Conference on Reverse Engineering*, pp.296–297, Los Alamitos, Victoria, BC, Canada, Nov. 2003.
- [22] K. Kuipers, "BNB20 solves mixed integer nonlinear optimization problems," <http://www.mathworks.com/matlabcentral/fileexchange/95-bnb>, Aug. 2003.
- [23] Micron Technology, Inc. "MT48H8M16LF Mobile SDRAM". 2003.
- [24] A. Shrivastava, I. Issenin, and N. Dutt, "Compilation techniques for energy reduction in horizontally partitioned cache architectures," *International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pp.90–96, New York, NY, USA, Sept. 2005.
- [25] C. Lee, M. Potkonjak, and W.H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," *Proc. 30th annual ACM/IEEE International Symposium on Microarchitecture*, pp.330–335, Research Triangle Park, NC, USA, Dec. 1997.
- [26] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," 2001. <http://www.eecs.umich.edu/jrjringenb/mibench/>.
- [27] "ARM926EJ-S Jazelle-enhanced macrocell". <http://www.arm.com/products/CPUs/ARM926EJ-S.html>, 2002.
- [28] P. Shivakumar and N.P. Jouppi, "CACTI 6.5: An Integrated Cache Timing, Power, and Area Model," <http://www.hpl.hp.com/research/cacti/>.
- [29] M. Absar, F. Polletti, P. Marchal, F. Catthoor, and L. Benini, "Fast and power-efficient dynamic data-Layout with DMA-capable memories," *1st Int'l Workshop on Power-Aware Real-Time Computer*, pp.1–4, Jan. 2004.
- [30] T.R. Mück and A.A. Fröhlich, "A run-time memory management approach for scratch-pad-based embedded systems," *Proc. ETFA*, pp.1–4, Sept. 2010.
- [31] R. Pyka, C. Faßbach, M. Verma, H. Falk, and P. Marwedel, "Operating system integrated energy aware scratchpad allocation strategies for multi-process applications," *Proc. SCOPES 2007*, pp.41–50, 2007.
- [32] S. Steinke, N. Grunwald, L. Wehmeyer, R. Banakar, M. Balakrishnan, and P. Marwedel, "Reducing energy consumption by dynamic copying of instructions onto onchip memory," *Proc. 15th International Symposium on System Synthesis*, pp.213–218, Kyoto, Japan, Oct. 2002.
- [33] The Intel IXP Network Processor. [DB/OL]. <http://developer.intel.com/technology/itj/2002/volume06issue03/>, 2002.
- [34] Freescale i.MX27X Multimedia Processors. [DB/OL]. <http://www.freescale.com/webapp/search/Serp.jsp>, 2008.

Jianping Wu received the B.S., degrees in Electrical Engineering and Automation from Xi'an Jiaotong University in 2001, M.S. degrees in IC Design from Southeast University in 2008, where he is currently pursuing the Ph.D. degree in Electronic Science and Technology. His research interests mainly include SoC memory sub-system and Optimization.

Ming Ling received the B.S., M.S., and Ph.D. degrees from Southeast University, Nanjing, China, in 1994, 2001, and 2011, respectively, all in Microelectronics & solid electronics. He is currently associate professor of SoC department of National ASIC system Engineering Research Center (CNASIC), Southeast University. His research interests include SoC memory subsystem, SoC Design method technology and Embedded Systems.

Yang Zhang received the B.S. degrees in Electrical Engineering from Nanjing University of Aeronautics and Astronautics, M.S. degrees in IC Design from Southeast University in 2006, 2009 respectively, where he is currently pursuing the Ph.D. degree in Microelectronics & solid electronics. His research interests mainly include SoC memory subsystem, SoC Design method technology.

Chen Mei received B.S. degree of electronics engineering from Anhui University in 2007 and M.S. degree of Circuits & Systems from Southeast University in 2010, China. Now, she is working toward the Ph.D. degree in Southeast University. Her research mainly focuses on re-configurable architecture design, memory optimization of SoC.

Huan Wang received B.S. degree of electronics engineering from Southeast University in 2008 and M.S. degree of Micro electronics and solid electronics from Southeast University in 2011, China. His research mainly focuses on memory optimization of SoC.