Power-aware Scheduling for Parallel Security Processors with Analytical Models*

Yung-Chia Lin Yi-Ping You Chung-Wen Huang
Jenq-Kuen Lee Wei-Kuan Shih Ting-Ting Hwang

National Tsing Hua University
Hsinchu 300 Taiwan

Abstract. Techniques to reduce power dissipation for embedded systems have recently come into sharp focus in the technology development. Among these techniques, dynamic voltage scaling (DVS), power gating (PG), and multiple-domain partitioning are regarded as effective schemes to reduce dynamic and static power. In this paper, we investigate the problem of power-aware scheduling tasks running on a scalable encryption processor, which is equipped with heterogenous distributed SOC designs and needs the effective integration of the elements of DVS, PG, and the scheduling for correlations of multiple domain resources. We propose a novel heuristic that integrates the utilization of DVS and PG and increases the total energy-saving. Furthermore, we propose an analytic model approach to make an estimate about its performance and energy requirements between different components in systems. These proposed techniques are essential and needed to perform DVS and PG on multiple domain resources which are of correlations. Experiments are done in the prototypical environments for our security processors and the results show that significant energy reductions can be achieved by our algorithms.

1 Introduction

Techniques to reduce power dissipation for embedded systems have recently come into sharp focus in the technology development, as power consumption has become a crucial issue in the embedded SOC design. Among these techniques, dynamic voltage scaling (DVS), power gating (PG), and multiple-domain partitioning are regarded as effective schemes to reduce both dynamic and static power. Dynamic voltage scaling [36] is to reduce the dynamic power consumption P by dynamically scaling the supply voltage V_{dd} and corresponding frequency f of the PE if no demands for full throttle operating. The DVS uses the following equations for architecture-level power estimation:

$$P_{dynamic} = C \times \alpha \times f \times V_{dd}^2$$
$$f = k \times (V_{dd} - V_t)^2 / V_{dd}$$

* This paper is submitted to *LCPC 2004*. The correspondence author of this work is Jenq Kuen Lee. His postal address is Prof. Jenq- Kuen Lee, Department of Computer Science, National Tsing-Hua Univ., Hsinchu, Taiwan. His e-mail address is jklee@cs.nthu.edu.tw, FAX number is 886-3-5723694, and phone number is 886-3-5715131 EXT. 3519.

where C is the switching capacitance, α is the switching activity, k is a circuit-dependent constant, and V_t denotes the threshold voltage. In the aspect of power gating [3, 26], the technique features in reducing leakage power dissipation; it uses the sleep circuit to disconnect the supply power from portions of the circuit when those portions are inactive. For leakage power estimation at architecture, we use the equation below:

$$P_{static} = V_{dd} \times N \times k_{design} \times I_{leakage}$$

where N is the number of transistors, k_{design} is a design-dependent constant, and $I_{leakage}$ denotes the normalized leakage current which depends on silicon technology, threshold voltage, and sub-threshold swing parameter. In the aspect of multiple domain partitioning, research issues remain for explorations when we try to integrate DVS, PG, and the scheduling for correlations of multiple domain resources. In this paper, we will have a scalable security processor as a case study to illustrate how to address this problem.

Variable Voltage Scheduling manages the tasks with execution deadlines and reduces power consumption without any task missing its deadline. We list the related research work as follows. The work in [10] gives a heuristic non-preemptive scheduling algorithm for independent tasks on a single processor. Works merely targeting on a single processor can be found in [8, 13, 14, 16, 23–25, 27, 31, 37, 38]. The works with distributed embedded systems can be found in [20–22, 29]. The main problem of using PG technology is to issue Gate-Off/On control at the proper time as to minimize performance degradation and maximize leakage power reduction. The works related to software-controlled power gating can be found in [6, 28, 39].

In this paper, we address the issue of variable voltage static scheduling in a heterogeneous distributed embedded system for independent periodic tasks with considering power gating to minimize both dynamic and static power dissipation. Our testbed is based on a scalable security processor (**SP**) which is developed in a collaborative research with the VLSI design group in our university [11, 17, 18, 33–35]. In the project, it aims to offer a configurable prototype of high-performance low-power security processors and incorporates the dynamic voltage scaling, power gating technology, and multiple domain partitioning in the designed processors. The interior of the security processors employs the architecture of heterogeneous distributed embedded system, in which processing elements (PEs) are various crypto-engine modules. Each crypto-engine module is designed to have DVS and PG capabilities. We propose a novel heuristic that integrates the utilization of DVS and PG, and increases the total energy-saving. Furthermore, we propose an analytic model approach to make an estimate about its performance and energy requirements between different components in systems. These proposed techniques are essential and needed to perform DVS and PG on multiple domain resources which are of correlation. Experiments are done in the prototypical environments for our security processors and the results show that significant energy reductions can be achieved by our proposed mechanisms.

The remainder of this paper is organized as follows. We first describe the architecture used in our target platform and power management design in Section 2. Next, we explain in detail our joint power-aware scheduling approach with both dynamic power reduction and leakage power reduction in Section 3. In Section 4, we present the experimental setup and the results. At last, we make an overall conclusion in Section 5.

2 Configurable SP Architecture with Power Management

In this section, we briefly describe a configurable architecture of **SP**s. The variations of this architecture have been used by many network device manufacturers, as Broadcom, Hifn [9], Motorola [7], and SafeNet. Key cryptographic functions in security processors may include:

- data encryption (DES, 3DES, RC4, and AES),
- user authentication only (DSS and DSA),
- hash function (SHA-1, MD5, and HMAC),
- public key encryption (RSA and ECC),
- public key exchange (Diffie-Hellman Key Exchange),
- and the compression (LZS and Deflate).

Fig.1 presents our architecture. It consists of a main controller, a DMA module, internal buses, and crypto modules [11, 17, 18, 33–35]. The main controller has a slave interface of external bus which accepts the control signals and returns the operation feedbacks via the interrupt port. In the main controller, there are resource allocation modules for managing resource such as external bus master interfaces, channels, transfer engines, internal buses, and crypto modules. Also, the process scheduler module and power management module are added into the main controller for task scheduling and low power control. The crypto operations are based on descriptors. The descriptor contains the type of en-/de-cryption functions, the length of raw data, the key information, and the destination memory address of the output data.

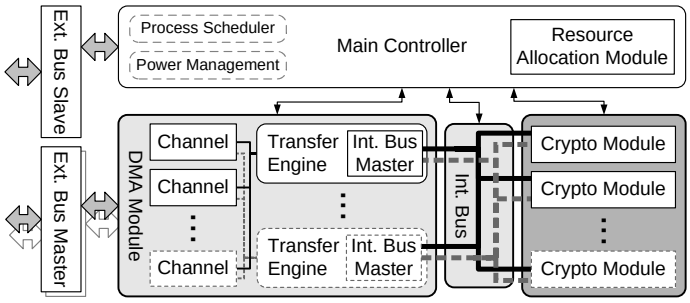

Fig. 1. Security processor architecture

The DMA module integrates master interfaces of external bus with the channels and the transfer engines. According to the data information in the channel, the transfer engine requests the external bus to transfer data from memory. The memory can be replaced by package processors which directly connect to the network by MAC modules. The transfer engine then passes the data to the dedicated crypto module via the internal bus. Furthermore, the internal buses are designed to support multiple layers for

high speed data transmission. Because the execution time of the crypto module may be varied, the crypto module will signal the main controller when the operations are done.

The power management module in the main controller presents a software-controllable voltage/speed adjustment of all components in the security processor. All components controlled by the power management module have four main power states: *Full*(1.8V), *Low*(1.5V), *Ultralow*(1.2V), and *Sleep*(0V). Cooperating with the process scheduler module, tasks can be assigned power states among *Full-Low-Ultralow*, and power gating as *Sleep* mode. For supplying multiple operating voltages, we have dynamic voltage generators which generate three level supply voltages for the security processor.

3 Power-aware Scheduling Approach

In this section, we discuss the scheduling issues for low power in our security architectures and focus on the problem of independant periodic task scheduling. We assume a distributed embedded system containing major PEs (crypto modules) which are capable of *k*-level supply voltages and power gating mode. Moreover, we assume the other non-PE components (such as buses, channels) are capable of DVS. To simplify scheduling problems with all components in a complicated system, we propose a three-phase iterative scheme for power-aware scheduling in the following:

1. We employ the scheduling methods in Section 3.1 and assume the maximum performance of non-PEs (such as bus and channels) to determine running voltage/frequency of each task in major PEs and appropriate power gating occasions.
2. Apply analytic approximation techniques to rapidly determine running voltages/frequencies of the remaining components in the system. Analytic methods also give the proper estimation of computation latency in PEs caused by these components in the system.
3. Re-employ the scheduling methods with information generated in phase 2 and deliver final scheduled setting of each task. Iteratively proceed with phases 2–3 till the scheduling results are invariable. Other details would be described in Section 3.2.

3.1 Joint Variable-Voltage Scheduling with Power Gating for PEs

It is known that the scheduling problem of a set of nonpreemptable independent tasks with arbitrary execution time on fixed-voltage multiprocessors is NP-complete [32]. With reduction techniques, we find that the same scheduling problem on variable-voltage multiprocessors is NP-hard. To optimize the power or energy consumption in real-time systems, we propose a heuristic algorithm to schedule tasks under the condition of variable supply voltage.

The proposed scheduler maintains a list, called the *reservation list* [30], in which these tasks are sorted by deadlines. Since each periodic task arrives with a certain periodicity, we can get the information about the arrivals and deadlines of tasks in a given interval. At the beginning, all tasks are in the list and sorted by their deadlines, and the task with the earliest deadline is then picked for scheduling. The scheduler first checks if the task could be executed completely prior to the deadline at the lower voltage without influencing any unscheduled task in the reservation list. In this way the scheduler

will decide how to schedule tasks at the lowest voltage as possible. For idle time of PEs, the scheduler will decide if it is deserved to gate off the idle PEs. The proposed heuristic would turn off unnecessary PEs completely as many as possible, so as to maximize static power reduction on top of dynamic power reduction.

Slack-Time Computation We first give the definition for the slack time in the scheduling. Suppose we are going to schedule task T_i, and there are still $(n-i)$ unscheduled tasks (i.e., T_{i+1}, T_{i+2}, ..., T_n) in the reservation list. The slack time $\delta_i(V)$ is the maximum period allowed for T_i while the remaining $(n-1)$ tasks are scheduled at supply voltage V in reverse order. To obtain the information for T_i, we first build a pseudo scheduler for the $(n-i)$ tasks with the following behaviors. The $(n-i)$ tasks are scheduled in a reversed way, which treats the deadlines as arrivals and the arrivals as deadlines and starts from the point of the latest deadline (i.e., d_n is the deadline of T_n) via the well-known earliest deadline first (EDF) algorithm [19]. We then record the time of the end point of the pseudo schedule as $\lambda_i(V)$.

The slack time of the pseudo schedule at a supply voltage V can be obtained from the following equation:

$$\delta_i(V) = \lambda_i(V) - Max(a_i, f_{i-1}),$$

where a_i is the arrival time of T_i, f_{i-1} is the finishing time of the last task T_{i-1}, and $Max(a,b)$ is a function that returns the maximum value of a and b. Figure 2 gives an example of the slack-time computation, in which there are four tasks in the reservation list. Here two reservation lists are maintained: one is created by a pseudo scheduler to schedule tasks at the lowest voltage, and the other is compiled by the highest-voltage scheduler. The slack time $\delta_i(V_H)$ and $\delta_i(V_L)$ is the time from the finishing time of the last task to the end point of the reservation list from the highest- and lowest-voltage schedulers, respectively. If we consider the overhead of DVS, the highest-voltage scheduler should add the maximum time-overhead of DVS to f_{i-1} to compute $\delta_i(V_H)$. It is noted that during the scheduling, we shall flag an exception if any deadline cannot be met when scheduling at the highest voltage.

Scheduling Algorithm The proposed scheduling algorithm is based on the EDF algorithm [19]. Figure 3 lists the algorithm. Assume there are n periodic tasks to be scheduled. First, we sort tasks in ascending order by deadlines, namely T_1, T_2, ..., T_n, and put them in a list of unscheduled tasks, i.e., the reservation list. We then extract each task from the list on the basis of the schedule. Suppose the system provides m processing elements, and each processing element is capable of K-level supply voltages, where level 1 represents the lowest voltage and level K represents the highest voltage. Steps 1–3 in Figure 3 describe these procedures. For utilizing power gating capabilities, we shall try to make tasks run successively without intermissions and let idle time be together because PG mechanisms cost much more expense than DVS does in performance and power. Next, in step 4, we compute the slack time for task T_i with both the highest- and lowest-voltage pseudo schedulers, denoted as $\delta_i(V_H)$ and $\delta_i(V_L)$. The slack time $\delta_i(V)$ represents the maximum time interval allowed for task T_i to execute while all the remaining tasks in the reservation list are scheduled in reverse order with supply

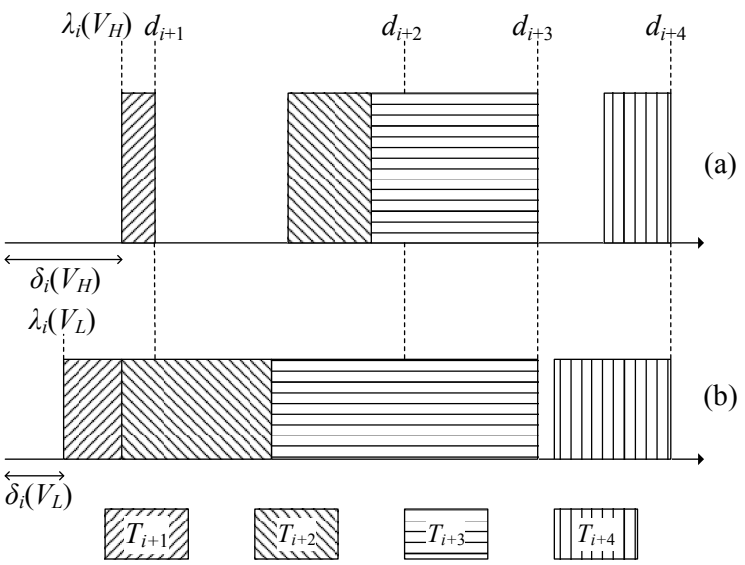

Fig. 2. Examples of slack-time computation while scheduling T_i: (a) tasks performed at the highest voltage; (b) tasks performed at the lowest voltage.

voltage V. In step 5, we compute the computation time of task T_i at both the highest- and lowest-voltages, denoted as $c_i(V_H)$ and $c_i(V_L)$. In step 6, we compare $c_i(V_H)$ and $c_i(V_L)$ with $\delta_i(V_L)$ and $\delta_i(V_H)$ to decide which voltage should be applied to the task. This algorithm results in three possible scenarios as follows:

(a) $c_i(V_L)$ plus time-overhead of voltage-scaling is smaller than or equal to $\delta_i(V_L)$. If energy-overhead of voltage-scaling is less than energy-saving, we can schedule task T_i at the lowest voltage without affecting any task in the future because there are no overlaps between task T_i and the unscheduled tasks while those tasks are assumed to be executed at the lowest voltage.

(b) $c_i(V_L)$ plus time-overhead of voltage-scaling is larger than $\delta_i(V_L)$ and smaller than or equal to $\delta_i(V_H)$. If this happens, we call a decision algorithm described in Section 3.1 to decide at which voltage task should T_i be scheduled. It weights the alternatives to optimize the overall costs, using a criterion such as the power or energy consumption.

(c) $c_i(V_L)$ plus time-overhead of voltage-scaling is larger than $\delta_i(V_H)$. This means it is impossible for task T_i to complete its execution by its deadline at any voltage lower than the highest voltage, and hence we must schedule it at the highest voltage to let its deadline be met. If task T_i is un-schedulable for current PE, we put it in a new list called L_{un} that contains all un-schedulable tasks.

In step 7, we check the remained idle time between the scheduled tasks in the current PE and determine power gating commands to be inserted if it benefits energy-saving. In step 8 and step 9, if the list L_{un} generated in step 6 is not empty, we use the list as the reservation list for the next available PE and schedule it by the same procedures in steps

Real-Time Scheduling Algorithm with Variable-Voltage Reservation Lists in Multiple Processing Elements

Input: n unscheduled periodic tasks and m PEs
Output: Schedule of gating commands and the n tasks with variable supply voltages at $PE_{1...m}$

1. Sort tasks by deadlines in ascending order; i.e., T_1, T_2, ..., T_n.
2. Put them in a list, called the *reservation list* of the target processing element PE_j. In the beginning, $_j$=1.
3. Remove the first task, namely T_i, which has the earliest deadline, from the *reservation list*. Repeat steps 3–6 while the list is not empty.
4. Compute the slack time for task T_i with both the highest- and lowest-voltage pseudo schedulers, i.e., $\delta_i(V_H)$ and $\delta_i(V_L)$.
5. Compute the computation time of T_i at the highest- and lowest-voltages, i.e., $c_i(V_H)$ and $c_i(V_L)$.
6. Let $o_t(i)$ be the voltage scaling time, schedule T_i in the following rules:
   - If $c_i(V_L)+o_t(i) \le \delta_i(V_L)$, schedule T_i for PE_j at V_L if possible†.
   - If $\delta_i(V_L) < c_i(V_L)+o_t(i) \le \delta_i(V_H)$, call the *decision algorithm*.
   - If $c_i(V_L)+o_t(i) > \delta_i(V_H)$ and
     - if $c_i(V_H)+o_t(i) \le \delta_i(V_H)$, schedule T_i for PE_j at V_H.
     - if $c_i(V_H)+o_t(i) > \delta_i(V_H)$, put T_i in an unscheduled list L_{un}.
7. Check idle-time of PE_j and insert gating commands if possible‡.
8. If PE_j is the last available PE and the list L_{un} is not empty, then report the possible failure of real-time scheduling.
9. If the list L_{un} is not empty, let $_j = _j + 1$ and use the list L_{un} as the *reservation list* of the target PE_j. Next, go to step 3.
10. If $j < m$, then gate off $PE_{j+1} \ldots PE_m$ all the time.

†Schedule T_i at V_L if deadline is met and energy-overhead is acceptable.
‡Gate on/off if tasks are unaffected and energy-overhead is acceptable.

Fig. 3. Reservation-list scheduling algorithm for variable-voltage problems in multiple PEs.

3–6. If no PE is available for scheduling, the scheduler should report the failure. At the last step, we will turn off all unused PEs via power-gating to minimize both static and dynamic power savings.

Decision Algorithm Following the notations in the previous subsections, assume that we are scheduling task T_i, and that$c_i(V_L)+o_t(i)$ is larger than $\delta_i(V_L)$ and smaller than or equal to $\delta_i(V_H)$. To achieve the objective of power/energy reduction, we propose several algorithms to decide at which voltage tasks should be scheduled when weighting trade-offs between tasks. We use a probability density function,

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \text{ where } -\infty < x < \infty,$$

which defines the probability density function for the value X of a random observation from the population [15], to divide the population of a group into K equal parts in terms

of area under the distribution, and then schedule tasks at levels corresponding to the parts that the tasks belong to. In other words, let W^1, W^2, ..., and W^{K-1} be demarcation that separates the population into K parts ; a task will be scheduled at level t if its value falls between W^{t-1} and W^t. The detailed algorithms are described as follows:

(a) *Reservation list with first-come first-served scheduling*
Tasks are always scheduled at the lowest voltage as possible without missing deadlines. This algorithm does not apply a cost model to the decision.

(b) *Reservation list with average power consumption*
We use the switching activity α_i to select the voltage level for T_i. We schedule a task at level $(K-\tau+1)$ if $\int_{-\infty}^{W_\alpha^\tau} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(W_\alpha^\tau-\mu)^2}{2\sigma^2}} = \frac{\tau}{K}$ and W_α^τ denotes the τ-th watershed of the population of switching activities of tasks.

3.2 Voltage/Speed Selection of Non-PEs

For non-PE components (such as buses and transfer engines) in the system, we apply an analytic modeling techniques (to be described shortly) to compute the suitable voltage so that total performance of the system will fit the scheduling results of major PEs.

Suppose the system has multiple PEs that are labeled with an index in the range 1 to l. Several channels, which are labeled with an index in the range 1 to n, are built into the control unit for simultaneously accessing the PEs. The data transfer between channels and PEs are across a few of internal buses, which are labeled with an index in the range 1 to m. We can view each of the k^{th} PE and the j^{th} internal bus as a server with a constant service rate of M_{s_k} and M'_{s_j} bits per second, respectively. Let $P_{i,k}$ be the probability that channel i makes its next service request to the PE k. Define Φ_i to be the average fraction of the time that the i^{th} channel is not waiting for a service request to be completed from any of PEs and internal buses. Also, let $\Omega_{k,i}$ and $\Omega'_{j,i}$ be the fraction of the time spent by the i^{th}channel waiting for a service request to PE k and internal bus j, respectively. Define $\frac{1}{M_{r_{k,i}}}$ to be the ratio of the time that descriptors of the i^{th} channel spend doing the overhead of PE service requests, not including the time of waiting in queues and having requests serviced by the k^{th} PE to the total processing data size. Let

$$\eta_k = \sum_{i=1}^{n} P_{i,k} \frac{M_{r_{k,i}}}{M_{s_k}} \quad , \quad \lambda_k = \frac{(1+\varepsilon_k) M_{s_k}}{\sum_{j=1}^{m} M'_{s_j}}$$

, where ε_k is the average scaling ratio of data size throughout the k^{th} PE processing. The average time that each channel spends doing initiating, host memory communication, and descriptor processing Φ_i is related to the time spent waiting, $\Omega_{k,i}$ *and* $\Omega'_{j,i}$, as the

following equations.

$$\Phi_i + \sum_{k=1}^{l} \Omega_{k,i} + \sum_{j=1}^{m} \Omega'_{j,i} = 1$$

$$\prod_{i=1}^{n} (1 - \Omega_{k,i}) + \eta_k \Phi_i = 1$$

$$\prod_{i=1}^{n} (1 - \Omega'_{j,i}) + \sum_{k=1}^{l} \eta_k \lambda_k \Phi_i = 1$$

We abbreviate the detailed model construction and proof which could be found in the appendix of this paper.

Suppose tasks for l PEs are scheduled by the scheduler as described in Section 3.1. We can derive Φ_i, $\Omega_{k,i}$, and $\Omega'_{j,i}$, which are the metrics of expected performance, from the scheduling results: the scheduling results offer the average service rate M_{s_k} of k^{th} processing engines and $\eta_k \Phi_i$, which semantically equal to utilization of k^{th} PE due to task assignments. Assume the security processor has n channels and m internal buses connecting PEs, so we can provide one proper selection among frequencies and corresponding voltages of internal buses and transfer engines through solving the equations previously: we use expecting values of M'_{s_j} to evaluate resulting Φ_i, $\Omega_{k,i}$, and $\Omega'_{j,i}$ and choose a minimal M'_{s_j} that causes the system to have the most load efficiency, for which Φ_i, $\Omega_{k,i}$, and $\Omega'_{j,i}$ are all positive and Φ_i should be the minimum as possible.

Besides voltage/frequency selection, the proposed analytic modeling is used to revise the latency parameters in the schedulers during scheme phases 2–3. In the realistic environments of considered systems, the computation time of tasks in PEs should actually include the latency time caused by data transmission and bus contention. The data transmission latency could be calculated validly by data size, bus speed, and detailed transfer operations during scheduling. The bus contention latency, however, could not be correctly estimated if lacking any runtime information. Thus we shall use the worst case estimation of the latency time to proceed the calculation of task computation time and avoid deadline missing under any runtime condition. In the first phase of scheduling scheme, we use maximum performance settings of internal buses and transfer engines, which relax the slack time computation, to schedule tasks. We conservatively assume the worst case of each task is waiting for all tasks in PEs except the one in which it is scheduled to complete their data transmission with the maximum time spent by the possible largest data transmission. The proposed analytic approximation phase of voltage-selection estimates possible low power settings of internal buses and transfer engines which match the scheduling results, and is also able to estimate more accurate worst-case latency time in each PE than theoretical one by means of $\Omega'_{j,i}$ and $\eta_k \Phi_i$, which would reflect the possible worst-case latency in the scheduling results. We then perform the third phase of scheduling scheme that uses values derived by the second phase, and obtain final scheduling results. Due to the monotonic property of PE usage in our scheduling algorithms, iterative processing of phases 2–3 would converge on a stable scheduling result.

4 Simulator & Experiment

For experiments, we have built a cycle-accurate simulator with energy models for configurable security processors. The cycle-accurate simulator is written in SystemC in which the simulation of each PE can be operated at assigned frequencies based on the voltage scaling. The energy consumption model, according to activity types and structure features of the hardware designs [4], is separated into functional units, control logics, memory, and clock distribution trees. Therefore, the energy consumptions are weighted with various considerations. The simulation environment needs operating with a power library which contains synthesis values of PEs under UMC 0.18 CMOS library.

suite	1	2	3	4	5	6	7	8	9
arrival distribution	uniform			normal			exponential		
job number	300								
jobs/time (μs)	1500		375	1500		375	1500		375
AES:RSA	30:1								
max data size (bytes)	1280								
max AES deadline (μs)	3072	3430		3072	3430		3072	3430	
max RSA deadline (μs)	13312	15872		13312	15872		13312	15872	
dynamic energy reduction (%)	19.41	20.09	15.95	20.50	18.56	28.64	21.13	31.15	33.30
leakage energy reduction (%)	83.71	82.92	83.39	83.17	83.85	83.55	82.40	83.24	83.29
total energy reduction (%)	19.41	20.09	15.96	20.50	18.56	28.64	21.13	31.15	33.30

Table 1. Benchmark settings and results

We implement a randomized security task generator to generate benchmark descriptor files for the simulator. The generator can generate the simulated OS-level jobs of de-/en-cryption and each job has randomized operation types, randomized data sizes, randomized keys and content, randomized arrival time, and randomized deadline, on the basis of an adjustable configuration of job arrival distribution types, job numbers, job density, ratio of distinct operation types, job size variance, and job deadline variance. Each generated job is then converted by the generator to the corresponding descriptors which can be executed by the simulator. In our preliminary experiments, we assume the SP has the architecture configuration of 6 AES modules, 2 RSA modules, 5 internal buses, 8 channels and transfer engines. The generated benchmarks consist of 9 test

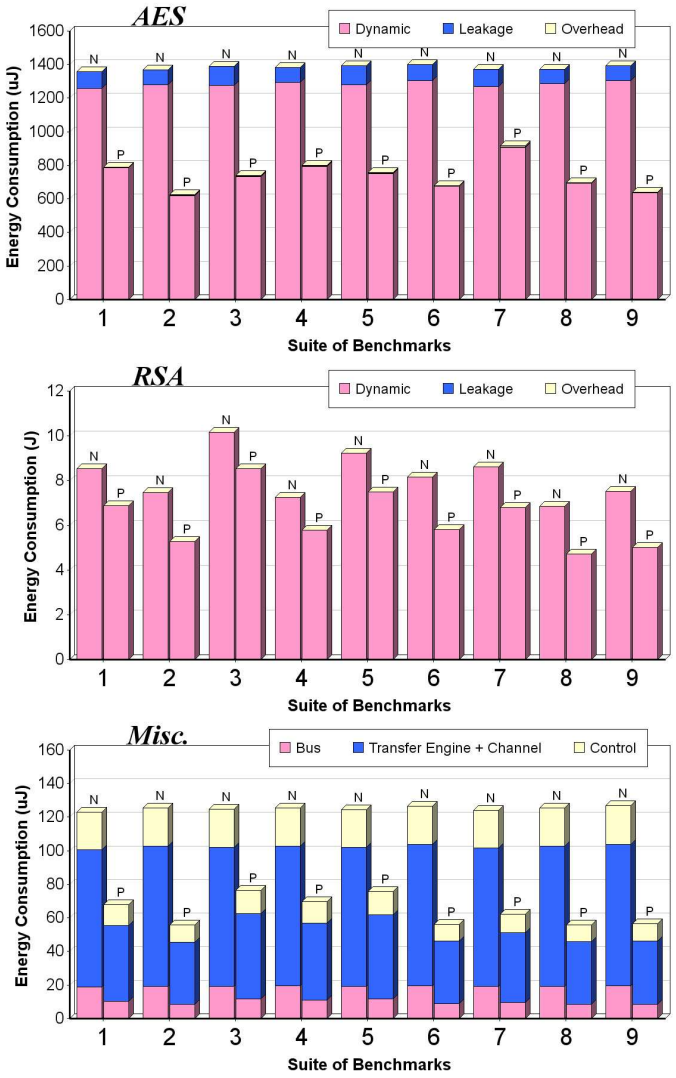

Fig. 4. Energy consumption estimated by the simulator

suites with different task generator configurations listed in Table 1. They are mainly divided into three types of arrival distributions. Each distribution type has three suites with different task slackness, which are dependent on job density and job deadline range: the first suite features high density and short deadline; the second one features high density and long deadline; the third one features low density and long deadline.

We have generated 100 distinct descriptor files for each suite and computed their average energy consumptions of different components from the results of the simulator, as shown in Figure 4. The bars labeled by **N** are the scheduling results without power

management and others labeled by **P** are the results with enabling our proposed power management. The energy-overhead of applying DVS and PG is too few to be exhibited clearly on the charts, and so does the leakage of RSA modules. The top chart gives the energy reduction for AES modules, the middle chart gives the energy reduction for RSA modules, and the bottom chart shows the energy reduction for non-PE components which are assigned by our analytic approximation phase, for all benchmark suites. The final results with latency approximation are also confirmed by the simulator that no deadline missing is reported in all benchmarks. Although the most energy consumptions are dominated by RSA operations in our experimental architecture and workloads, the charts show that our scheme performs well for all components in the system. Moreover, the leakage reduction is great as shown in Table 1 and this portion is expecting to be more important when CMOS process is going under $0.13\mu m$ [1, 5]. Table 1 also gives the overall energy reduction for all test suites. The summary in the table presents that the total energy reduction up to 33% could be achieved by our power-aware scheduling scheme.

5 Conclusions

In this paper, we present a new approach of increasing power efficiency in complex distributed embedded systems with dynamic and static power reduction mechanisms. As the preliminary results show, our power management scheme gets significant power reduction for the experimental security processor. This work gives an exploration study for variable voltage scheduling resources of multiple domains with correlations.

References

1. International technology roadmap for semiconductors 2003 edition. Technical report, Semiconductor Industry Association.
2. F. Bodin, D. Windheiser, W. Jalby, D. Atapattu, M. Lee, and D. Gannon. Performance evaluation and prediction for parallel algorithms on the bbn gp1000. In *Proc. of the 4th ACM International Conference on Supercomputing*, pages 401–403, June 1990.
3. J. A. Butts and G. S. Sohi. A static power model for architects. In *Proc. Int. Symp. on Microarchitecture*, pages 191–201, December 2000.
4. R. Y. Chen and M. J. Irwin. Architecture-level power estimation and design experiments. *ACM Transactions on Design Automation of Electronic Systems*, 6(1):50–66, 2001.
5. B. Doyle, R. Arghavani, D. Barlage, S. Datta, M. Doczy, J. Kavalieros, A. Murthy, and R. Chau. Transistor elements for 30nm physical gate lengths and beyond. *Intel Technology Journal*, 6:42–54, May 2002.
6. D. Duarte, Y. Tsai, N. Vijaykrishnan, and M. J. Irwin. Evaluating run-time techniques for leakage power reduction. In *Proc. ASPDAC*, January 2002.
7. N. Gammage and G. Waters. *Securing the Smart Network with Motorola Security Processors*, March 2003.
8. F. Gruian and K. Kuchcinski. Lenes: Task scheduling for low-energy systems using variable supply voltage processor. In *Proc. ASPDAC*, pages 449–455, January 2001.
9. Hifn. *7954 security processor Data Sheet*, December 2003.

10. I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava. Power optimization of variable-voltage core-based systems. *IEEE Trans. Computer-Aided Design*, 18(12):1702–1714, December 1999.
11. J.-H. Hong and C.-W. Wu. Cellular array modular multiplier for the rsa public-key cryptosystem based on modified booth's algorithm. *IEEE Transactions on VLSI Systems*, 11:474–484, 2003.
12. K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing*. Mc Graw-Hill, 1984.
13. T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proc. ISLPED*, pages 197–202, August 1998.
14. C. M. Krishna and L.-H. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proc. Real Time Technology and Applications Symp.*, May 2000.
15. L. L. Lapin. *Modern Engineering Statistics*. Wadsworth Publishing Company, WBelmont, CA,, 1997.
16. C.-R. Lee, J.-K. Lee, T.-T. Hwang, and S.-C. Tsai. Compiler optimizations on vliw instruction scheduling for low power. *ACM Transactions on Design Automation of Electronic Systems*, 8(2):252–268, 2003.
17. M.-C. Lee, J.-R. Huang, C.-P. Su, T.-Y. Chang, C.-T. Huang, and C.-W. Wu. A true random generator desing. In *13th VLSI Design/CAD Symp.*, Auguest 2002.
18. T.-F. Lin, C.-P. Su, C.-T. Huang, and C.-W. Wu. A high-throughput low-cost aes cipher chip. In *3rd IEEE Asia-Pacific Conf. ASIC*, Auguest 2002.
19. C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.
20. J. Luo and N. Jha. Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In *Proc. ASPDAC*, January 2002.
21. J. Luo and N. K. Jha. Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems. In *Proc. ICCAD*, pages 357–364, November 2000.
22. J. Luo and N. K. Jha. Battery-aware static scheduling for distributed real-time embedded systems. In *Proc. DAC*, pages 444–449, June 2001.
23. A. Manzak and C. Chakrabarti. Variable voltage task scheduling for minimizing energy or minimizing power. In *Proc. ICASSP*, pages 3239–3242, 2000.
24. T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proc. ISLPED*, pages 76–81, August 1998.
25. J. Pouwelse, K. Langendoen, and H. Sips. Energy priority scheduling for variable voltage processors. In *Proc. ISLPED*, Augest 2001.
26. M. D. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar. Gated-vdd:a circuit technique to reduce leakage in deep-submicron cache memories. In *Proc. ISLPED*, 2000.
27. G. Quan and X. Hu. Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors. In *Proc. DAC*, pages 828–833, 2001.
28. S. Rele, S. Pande, S. Onder, and R. Gupta. Optimizing static power dissipation by functional units in superscalar processors. In *Proc. Int. Conf. on Compiler Construction*, pages 261–275, 2002.
29. M. T. Schmitz and B. M. Al-Hashimi. Considering power variations of dvs processing elements for energy minimisation in distributed systems. In *Proc. ISSS*, pages 250–255, October 2001.
30. W.-K. Shih and J. W. S. Liu. On-line scheduling of imprecise computations to minimize error. *SIAM Journal on Computing*, 25(5):1105–1121, 1996.
31. Y. Shin and K. Choi. Power conscious fixed priority scheduling for hard real-time systems. In *Proc. DAC*, pages 134–139, June 1999.

32. J. A. Stankovic, M. Spuri, M. D. Natale, and G. Buttazzo. *Implications of Classical Scheduling Results For Real-Time Systems*, volume 28. 1995.
33. C.-P. Su, T.-F. Lin, C.-T. Huang, and C.-W. Wu. A highly efficient aes cipher chip. In *ASP-DAC*, January 2003.
34. C.-Y. Su, S.-A. Hwang, P.-S. Chen, and C.-W. Wu. An improved montgomery algorithm for high-speed rsa public-key cryptos ystem. *IEEE Transactions on VLSI Systems*, 7:280–284, 1999.
35. M.-Y. Wang, C.-P. Su, C.-T. Huang, and C.-W. Wu. An hmac processor with integrated sha-1 and md5 algorithms. In *ASP-DAC*, January 2004.
36. M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced cpu energy. In *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 13–23, 1994.
37. F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced cpu energy. In *Symp. Foundations of Computer Science*, pages 374–382, 1995.
38. Y.-P. You, C.-R. Lee, and J.-K. Lee. Real-time task scheduling for dynamically variable voltage processors. In *Proc. IEEE Workshop on Power Management for Real-Time and Embedded Systems*, May 2001.
39. Y.-P. You, C.-R. Lee, and J.-K. Lee. Compiler analysis and support for leakage power reduction on microprocessors. In *Proc. LCPC*, July 2002.

Appendix - Analytical Models

We describe the analytic model developed for the security architectures given earlier in Section 2 as follows. We consider a typical execution process of an operation in the system described in Section 2. The execution of each operation can be viewed as a procedure that a channel requests an internal bus twice to serve the data transmission and requests a PE to serve the data manipulation. Assume each channel execution can be treated as an exponentially distributed random process which produces sets of service request with three correlated operations in the fixed order: two for the internal bus, one for the PE. Following the notations in Section 3.2, let $system\_cycles_i$ be the total time spent by the i^{th} channel on transmitting over system bus (including host memory accessing, descriptor processing, or idle). We now give the definition for $request\_cycles_i$. The $request\_cycles_i$ has two elements. It includes the total time spent by the i^{th} channel on preparing PE request and internal bus request and processing time. Moreover, it includes the total time that the data are traversing among the channel, internal buses, and PEs. Now let $channel\_cycles_i$ be

$$channel\_cycles_i = system\_cycles_i + request\_cycles_i$$

Define $M_{r_{k,i}}$

$$M_{r_{k,i}} = \frac{data\_amount_{k,i}}{channel\_cycles_i}$$

which is the ratio of data request amount to the time that descriptors of the i^{th} channel spend doing the overhead of PE service requests, not including the time of waiting in queues and having requests serviced by the k^{th} PE.

If we neglect the interaction between channels and assume that all internal buses are utilizable by all channels and PEs, then we have the following analytic model developed on top of the previous parallelizing theorem [2, 12]:

Theorem 1 *Let*

$$\eta_k = \sum_{i=1}^{n} P_{i,k} \frac{M_{r_{k,i}}}{M_{s_k}} \quad , \quad \lambda_k = \frac{(1+\varepsilon_k)M_{s_k}}{\sum_{j=1}^{m} M'_{s_j}}$$

, where ε_k is the average scaling ratio of data size throughout the k^{th} PE processing. The average time that each channel spends doing initiating, host memory communication, and descriptor processing Φ_i is related to the time spent waiting, $\Omega_{k,i}$ and $\Omega'_{j,i}$, as the following equations.

$$\Phi_i + \sum_{k=1}^{l} \Omega_{k,i} + \sum_{j=1}^{m} \Omega'_{j,i} = 1$$

$$\prod_{i=1}^{n}(1-\Omega_{k,i})+\eta_k\Phi_i=1$$

$$\prod_{i=1}^{n}(1-\Omega'_{j,i})+\sum_{k=1}^{l}\eta_k\lambda_k\Phi_i=1$$

Proof. The first equation simply infers the time conservation. Let $C_{k,i}$ be the average channel-i-to-PE-k request cycle time for the system, $total\_cycle$ be the total operation time per request, and we get

$$\frac{1}{C_{k,i}}=\frac{data\_amount_{k,i}}{total\_cycles} \tag{1}$$

on average. By observing the workloads, we can compute $M_{r_{k,i}}$ which is the ratio of the request data amount to channel cycles. Based on the definition of $M_{r_{k,i}}$ and equation (1), we can derive

$$\frac{1}{M_{r_{k,i}}C_{k,i}}=\frac{channel\_cycles_i}{total\_cycles}=\Phi_i \tag{2}$$

Moreover, define

$$\delta_{k,i}=\begin{cases}1 & if\ channel\ i\ is\ not\ waiting\ for\ module\ k\\ 0 & otherwise\end{cases}$$

$$\delta'_{j,i}=\begin{cases}1 & if\ channel\ i\ is\ not\ waiting\ for\ bus\ j\\ 0 & otherwise\end{cases}$$

Let μ_k be the probability that PE k is busy and μ'_j be the probability that internal bus j is busy. We have

$$\mu_k=1-E(\delta_{k,1}\delta_{k,2}\cdots\delta_{k,n}) \tag{3}$$

$$\mu'_j=1-E(\delta'_{j,1}\delta'_{j,2}\cdots\delta'_{j,n}) \tag{4}$$

where $E(\nu)$ is the expected value of the random variable ν. Therefore, $\mu_k M_{s_k}$ and $\mu'_j M'_{s_j}$ are the rate of completed requests to PE k and internal bus j, respectively. When the system is in equilibrium, $\mu_k M_{s_k}$ is equivalent to the rate of submitted requests to PE k and $\mu'_j M'_{s_j}$ is equivalent to the rate of submitted requests to internal bus j. Since $\sum_{i=1}^{n}\frac{P_{i,k}}{C_{k,i}}$ is the total rate of submitted requests to PE k from all channels,we have the equivalence,

$$\sum_{i=1}^{n}\frac{P_{i,k}}{C_{k,i}}=\mu_k M_{s_k} \tag{5}$$

Likewise, $\sum_{k=1}^{l}\sum_{i=1}^{n}\frac{P_{i,k}}{C_{k,i}}$ is the average rate of submitted requests to all internal buses from all channels. Due to the law of data indestructibility, we could have $\sum_{k=1}^{l}\sum_{i=1}^{n}\frac{P_{i,k}(1+\varepsilon_k)}{C_{k,i}}$ to be the average rate of submitted requests to all internal buses from all channels and all PEs. Accordingly, we have the equivalence as follows:

$$\sum_{k=1}^{l}\sum_{i=1}^{n}\frac{P_{i,k}(1+\varepsilon_k)}{C_{k,i}}=\sum_{j=1}^{m}\mu'_j M'_{s_j} \tag{6}$$

By combing equations (2), (3), (4), (5), and (6), we get

$$\begin{cases}\eta_k=\sum_{i=1}^{n}P_{i,k}\frac{M_{r_{k,i}}}{M_{s_k}}\\ E(\delta_{k,1}\delta_{k,2}\cdots\delta_{k,n})+\eta_k\Phi_i=1\end{cases},\qquad \begin{cases}\lambda_k=\frac{(1+\varepsilon_k)M_{s_k}}{\sum_{j=1}^{m}M'_{s_j}}\\ E(\delta'_{j,1}\delta'_{j,2}\cdots\delta'_{j,n})+\sum_{k=1}^{l}\eta_k\lambda_k\Phi_i=1\end{cases}$$

Nevertheless, since both $\delta_{k,i}$ and $\delta'_{j,i}$ are binaries, we have by symmetry

$$E(\delta_{k,i})=1-\Omega_{k,i}\quad,\quad E(\delta'_{j,i})=1-\Omega'_{j,i}$$

for each channel i. We now make a critical approximation by assuming that all the channels have non-correlated activities and we get

$$E(\delta_{k,1}\delta_{k,2}\cdots\delta_{k,n})=E(\delta_{k,1})E(\delta_{k,2})\cdots E(\delta_{k,n})=\prod_{i=1}^{n}(1-\Omega_{k,i})$$

$$E(\delta'_{j,1}\delta'_{j,2}\cdots\delta'_{j,n})=E(\delta'_{j,1})E(\delta'_{j,2})\cdots E(\delta'_{j,n})=\prod_{i=1}^{n}(1-\Omega'_{j,i})$$

The result follows.