Power Gating Aware Task Scheduling in MPSoC

Yu Wang, *Member, IEEE*, Jiang Xu, *Member, IEEE*, Yan Xu, Weichen Liu, *Student Member, IEEE*, and Huazhong Yang, *Senior Member, IEEE*

***Abstract*—Shrinking the feature size allows more and better functions on a single chip. However, it makes multiprocessor system-on-chip (MPSoC) more susceptible to various reliability threats. Power supply noise is a major reliability problem faced by low power MPSoCs using power gating techniques. Powering on and off a processing unit in MPSoCs will induce large power/ground (P/G) noise and can cause timing divergence and even functional errors in surrounding processing units. Previous work on resilient architectures mainly focused on power/thermal management and neglected the important side-effect: P/G noise induced by power gating. In this paper, for the first time, we formulate a task scheduling problem with the consideration of P/G noise based on our detailed P/G noise analysis platform for MPSoC. Two efficient algorithms are proposed to reduce noise protection penalty and improve MPSoC performance. Our experiments show that both simulated annealing and heuristic algorithms can achieve on average 25% performance improvement together with up to 80% noise protection penalty saving compared with the conservative stop-go method for short tasks (shorter than 20 K clock cycles). For longer tasks up to 200 K clock cycles, the performance improvement of our methods will become relatively low. However, we can still achieve at least 35.2% noise protection penalty saving. Furthermore, a lightweight online adjustment strategy accompanying the offline scheduling method is proposed to adapt to runtime variations and improve reliability.**

***Index Terms*—Mixed integer linear programming (MILP), multiprocessor system-on-chip (MPSoC), power gating, power ground (P/G) noise, task scheduling.**

I. Introduction

With the continuous development of semiconductor technologies, reliable system design becomes significantly more challenging as designers must contend with unreliable components and design processes. Although shrinking the feature size results in the improvement of functionalities, a multiprocessor system-on-chip (MPSoC) with smaller feature sizes is more susceptible to various reliability threats, such as silicon failures, noise effects, extreme process variation, design errors, and etc. Reliability-aware resilient design considerations or even resilient architectures are needed to protect MPSoC during the runtime.

Power supply noise is one of the most significant reliability threats for MPSoCs with smaller feature sizes. Strict low power requirements have led to the adoption of aggressive techniques such as dynamic voltage and frequency scaling, clock gating, and power gating [1], [2]. Although techniques like power gating can dramatically reduce power consumption for idle cores, they also exacerbate simultaneous switching noise (or ΔI noise) on the power delivery network. At the same time, when process technology advances, power consumption and wire resistance have gone up while the supply voltage drops. Thus the chip noise margin will go down. As a result, to design resilient systems, design methodologies with power supply noise management become necessary to fulfil the low power and high reliability requirements of MPSoCs.

MPSoCs use multiple system performance within their power budget. However, it is not likely or in some cases even impossible that all of these PUs are active at the same time. Power gating is a mature solution to eliminate the ever-increasing leakage power consumption. When a new task is assigned to a power-off PU or a PU finishes a task, the powering on/off will cause large noise in the power delivery network and then propagate to the PUs around the powering-on/off PU. For example, powering on a PU in the corner of a low power 16-PU-MPSoC, the peak noise level of this PU is over 30% of the supply voltage, and the noise level in the adjacent PU can achieve about 20% of the supply voltage (see Section III, Table I). Voltage variations in P/G network will bring delay variations to the internal circuits of affected PUs. Increased delay will lead to performance degradation, while decreased delay may introduce racing risk. Hence P/G noise will threaten the performance and reliability of circuits. In the open literature, there are few studies addressing such noise issues in MPSoCs. If a new task is assigned to a power-off PU, the active PUs around it need to be *protected* from the powering-on *ATTACK*; similarly, if a PU finishes a task, the active PUs around it also needs to be *protected* from the powering-off *ATTACK*. However, *protection* is not free: extra time and power are needed to clock gate the victim PUs and to wake up them when the attack PU is fully turned on or off.

In this paper, we focus on the P/G noise aware power gating strategies in MPSoCs: protecting MPSoCs that use the power gating technique from P/G noise, while minimizing the protection penalty. Our contributions are summarized as follows.

1) A detailed P/G noise analysis platform for MPSoC is proposed to model and study the P/G noise generation and propagation. The noise induced by powering on/off PUs in different positions, i.e., the spatial and temporal distribution of P/G noise induced by different PU behaviors, can be

Manuscript received August 26, 2009; revised February 20, 2010; accepted June 18, 2010.

Y. Wang, Y. Xu, and H. Yang are with the Circuits and Systems Division, Electronics Engineering Department, Tsinghua University, Beijing 100084, China (e-mail: yu-wang@mail.tsinghua.edu.cn; waterphy@gmail.com; yanghz@tsinghua.edu.cn).

J. Xu and W. Liu are with the Hong Kong University of Science and Technology, Hong Kong, China (e-mail: jiang.xu@ust.hk; weichen@ust.hk).

This work was supported in part by National Science and Technology Major Project, 2010ZX01030-001-001-04, by NSFC (60870001), by 863 Program of China (2009AA01Z130), by TNList Cross-discipline Foundation, by RGC of the Hong Kong Special Administrative Region, China (Funding 621108), and by HKUST PDF.

Digital Object Identifier 10.1109/TVLSI.2010.2055907

obtained from our platform. Meanwhile, according to the safe voltage level derived from the performance requirement of standard cells, the distinct *impact range* of powering on/off each PU is defined.

2) For the first time, the P/G noise aware power gating problem in MPSoC is formulated using mixed integer linear programming (MILP), where MPSoC performance and P/G noise protection overhead induced by power gating are optimized. The objective function of the MILP is to minimize: 1) the total execution time for all the tasks and 2) the performance overhead for protecting the victims from the power gating induced P/G noise (specifically, the number of clock gating and power on/off operations).
3) Two alternative offline task scheduling techniques are proposed based on our MPSoC P/G noise model considering both spacial and temporal constraints. One is based on Simulated Annealing algorithms, marked as SA, to obtain optimized scheduling results; the other is a fast heuristic algorithm, marked as HA, to achieve efficiency in terms of running time. The experiment results show that SA and HA can achieve up to 39.6% performance improvement and reduce 80% overhead for noise protection compared with the conventional stop-go method for tasks shorter than 20 K clock cycles. For longer tasks up to 200 K clock cycles, the performance improvement will become relatively lower. But we can still achieve at least 35.2% noise protection penalty saving. Our method can make a tradeoff between performance and noise protection overhead reduction. Study shows that a 20% relaxation in execution time can reduce up to 40% noise protection penalty.
4) Since task execution time may vary during the real-time operation, a lightweight online adjustment strategy is also proposed to assist the offline scheduling to improve the chip reliability.

The rest of this paper is organized as follows. Section II reviews the related work. P/G noise analysis platform for MPSoC is provided in Section III. In Section IV, we formulate the static scheduling problem in MPSoC using MILP. Section V describes two efficient algorithms to improve the speed of offline scheduling. Section VI describes the online adjustment strategy to handle exceptions of realtime execution time variations. Section VII shows the experimental results and related analysis. Section VIII concludes this paper and discusses possible future extensions.

II. Related Work

A. Power/Ground (P/G) Noise Mitigation

Previous work on P/G noise mitigation for power gating techniques mainly focused on circuit level techniques, such as sleep transistor designs [3]–[6] , decoupling capacitor insertion [7], and P/G noise aware floorplanning [1], [2], [8]. Optimum power gating sleep transistor design and implementation are critical to a successful low-power design. Various power-on current rush control methods through the sleep transistor implementation to reduce power supply voltage fluctuation were investigated in [3]–[6]. A low power switched decoupling capacitor circuit was proposed to suppress on-chip resonant supply noise [7]. Jiang [8] developed a power-gating driven floorplanner (PGFP) to assist in designing of power gated chips. Mohamood [1] proposed noise-direct, a design methodology for power integrity aware floorplanning, using micro-architectural feedback to guide the module placement. Healy [2] presented an improved design methodology to combat the ever-aggravating high frequency power supply noise (di/dt) in modern microprocessors based on Mohamood's work.

Recently, power gating sequence scheduling [9]–[11] in a block or several blocks was proposed to tradeoff between wake-up time and the P/G noise. The authors of [9] partition the circuit given the constraint that the maximum current is drawn from the power grid when the sleep transistor is switching, and they propose a polynomial-time algorithm to minimize the total wake-up time. In [10], Ramalingam *et al.* assume that each gate can be turned on at any time subject to the fan-in and current constraints and formulate the scheduling of sleep transistors' wake up times as a MILP problem. In [11], Jiang and Marek-Sadowska address system-level power gating with several gated blocks and optimize the wake up order of these blocks in terms of noise.

These works mainly focused on block-level design techniques, while in this paper, we investigate processor-level power gating-aware scheduling strategies based on our detailed P/G noise analysis platform for MPSoC, with the objective of minimizing performance degradation caused by noise protection during powering on/off PUs. Furthermore, our approach is based on circuit level simulations, which provides accurate P/G noise estimations to guide the search towards the optimal solutions.

B. Scheduling Problems

Many works have been done to address the scheduling problems with power/performance/thermal objectives [12]–[18]. However, P/G noise is different from thermal and energy, which have accumulative effects. The noise level should be predicted and victim circuits should be protected before the noise is induced. Hence, the power gating aware scheduling problem with the consideration of P/G noise should be carefully modeled and solved offline based on an accurate P/G noise estimation, and then assisted by a fast online adjustment method considering the run-time variations of tasks' execution times. Recent work by Todri [19] considered the P/G noise induced by switching current of active PUs to minimize the P/G noise level of multi-core systems. In this work, we mainly focus on modeling and management of the noise induced by powering-on/off a PU when a task is assigned to or finishes on it.

III. Analysis on Power Gating Induced P/G Noise in MPSoC

In this section, our power gating induced P/G noise analysis platform for MPSoC is introduced. The framework of the platform is illustrated in Fig. 1. There are three key components in the platform: 1) P/G network and package modeling of MPSoC; 2) MPSoC specification, such as PU number, layout, behavior, etc.; and 3) technology parameters, such as technology node related physical parameters. We use HSpice to simulate the P/G noise. We can obtain the noise levels induced by powering on/off

TABLE I
PEAK NOISE OF DIFFERENT PUS WHEN PU1 IS ATTACKER. D IS THE DISTANCE OF PUS (FROM CENTER OF ONE PU TO CENTER OF ANOTHER PU); N IS THE NOISE VALUE OF EACH PU

D	0	1	$\sqrt{2}$	2	$\sqrt{5}$	$\sqrt{8}$	3	$\sqrt{10}$	$\sqrt{13}$	$\sqrt{18}$
N(mv)	243	149	112	106	97	91	97	93	90	89

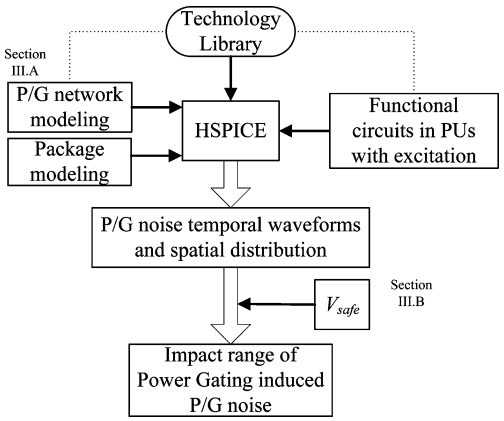

Fig. 1. Power gating induced P/G noise analysis platform.

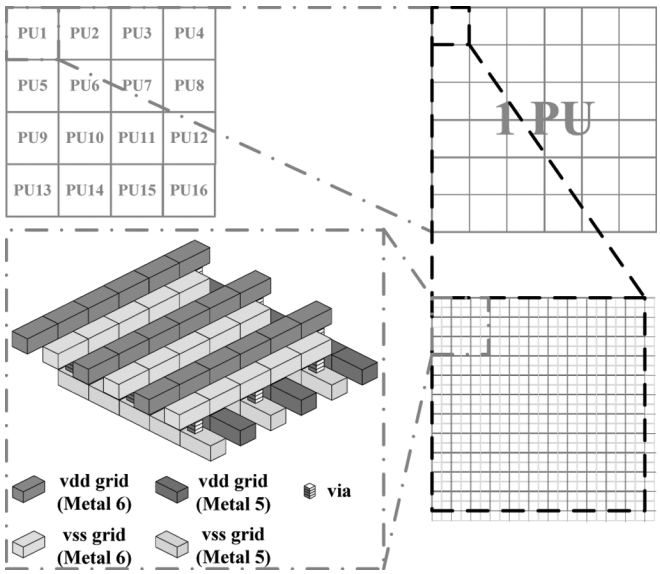

Fig. 2. P/G network architecture of MPSoC.

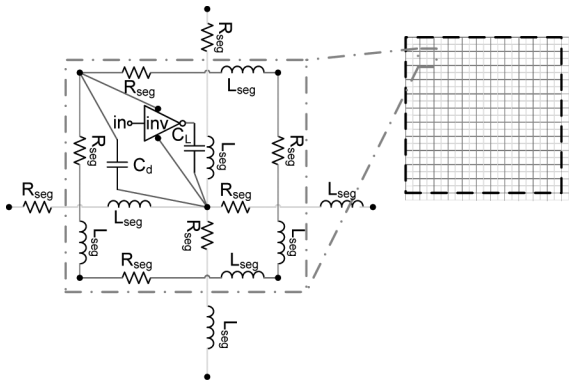

Fig. 3. In order to facilitate the P/G network analysis, each wire segment is modeled as a chain of L-type RLC equivalent circuits. An inverter with a capacitance load is used to imitate the core logic. A decap is connected to the intersection points on the vdd/vss grids.

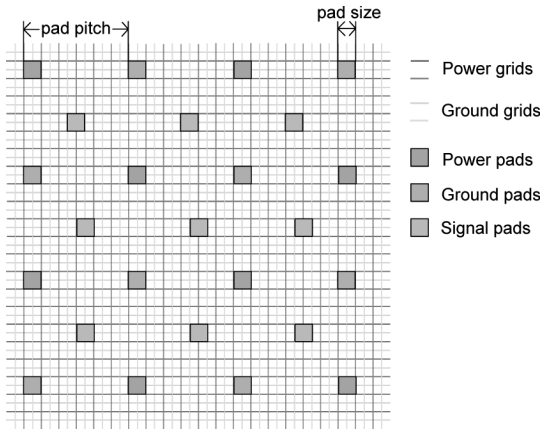

Fig. 4. On-chip P/G grids and I/O pads.

PUs in different positions and time instants, i.e., the spatial and temporal distribution of P/G noise induced by different PU behaviors. According to the safe voltage level derived from the performance requirement of standard cells, we obtain the *impact range* of a PU, which is defined as the set of neighboring PUs whose voltage level can be affected by power gating on it.

A. P/G Network Modeling of MPSoC

As shown in Fig. 2, the MPSoC chip is composed of PUs organized in 2-D-mesh topology.

1) P/G Network Model: As mesh-based power distribution networks design is the commonly accepted design approach to handle current variations [20] and meshes are the accepted way of designing power distribution networks in modern design [21], we assume the power rail distribution as mesh-based power/ground grid here. The most common way to distribute power in a gigascale integration (GSI) chip is to distribute it through an on-chip grid made of orthogonal segments (see Fig. 2) [22]. The horizontal and vertical segments of a grid are routed at different metal levels (e.g., at the layers of Metal 5 and Metal 6) and are connected through vias at the crossing points. The number of segments in the P/G grid is usually large. Neglecting the via resistance, each node of the power distribution grid is connected to the four neighboring nodes. A wire between two nodes is simply modeled as a lumped resistance R_{seg} and an inductance L_{seg} (see Fig. 3). C_d denotes the capacitance per unit area between a power grid node and the adjacent ground grid node (including both the intentionally added on-chip decaps and the equivalent capacitance between the wires at different metal levels). C_L is the load capacitance. l_{seg} is the length of the segments in horizontal and vertical directions. W_{seg} and T_{seg} are the width and thickness of the segments, respectively. We set the wire segment dimensions above, and the resistance-inductance-capacitance (*RLC*) parameters are calculated through the PTM interconnect model [23]. Both [24] and our simulation results show

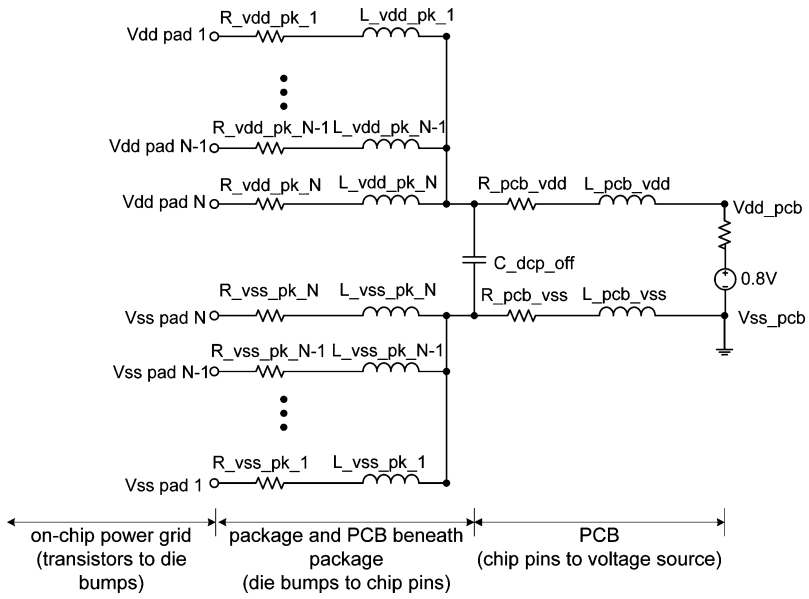

Fig. 5. Package model with package L and off-chip decoupling capacitance.

that on-die inductance has little impact to P/G noise. Hence the on-chip inductance L_{seg} is neglected during the simulation.

2) Package Model: In flip-chip package technology, the package input/output (I/O) pads are connected to the chip I/O pads through metal bumps distributed across the chip surface.

The flip-chip package is more expensive than a wire-bond package. However, it has smaller I/O parasitics [25]. The distributed pads also help increase total I/O count and consequently lower the P/G noise. Power distribution for a high performance microprocessor requires many pads. Two-thirds of the total pads are used for power distribution [26] in our model. These power and ground pads are spread throughout the chip surface, as shown in Fig. 4.

In our package model, as Fig. 5 shows, the pad and bump are modeled as a package resistance (R_vdd_pk_N and R_vss_pk_N) and a package inductance (L_vdd_pk_N and L_vss_pk_N). Besides, an off-chip decoupling capacitance (C_dcp_off) is added between the virtual power node and the virtual ground node of each core. The PCB board is modeled as a lumped resistance (R_pcb_vdd/vss) and inductance network (L_pcb_vdd/vss) here. It should be pointed out that the values of package resistance and package inductance are in direct proportion to the distance between the current pad and the center pad of the chip, which is in accord with the fact that the power supply usually locates in the center of the chip (e.g., our MPSoC). The package P/G network is included in the chip package and PCB beneath package in Fig. 5. We use this radial network as a simplified package model.

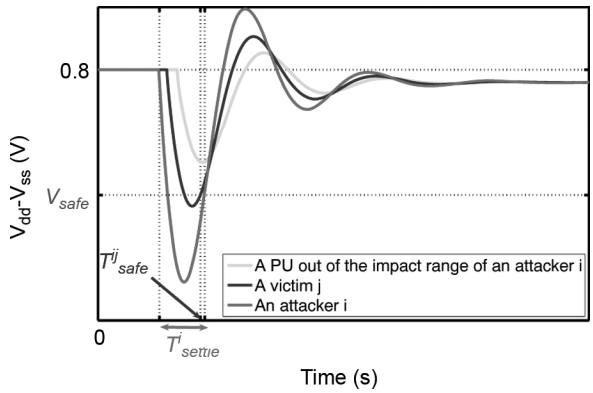

Fig. 6. Conceptual illustration of P/G noise temporal influence in MPSoC.

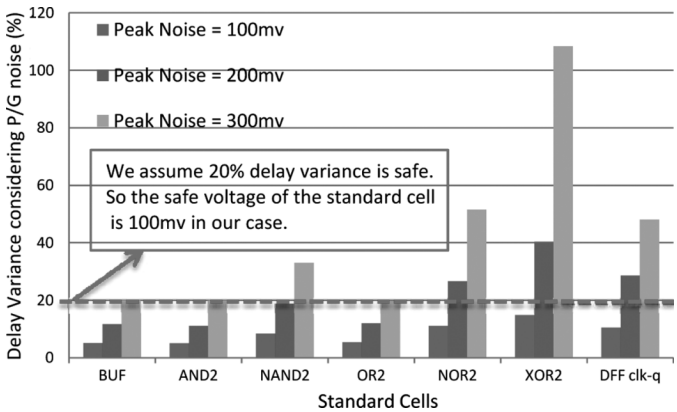

Fig. 7. Delay variance of standard cells considering P/G noise. We use 45 nm standard cell from [28].

B. Power Gating Induced P/G Noise Estimation in MPSoC

The power on and off of a PU will cause large transient current demands, which will induce large voltage deviations in the P/G grids. Based on the P/G model in the above subsection, the noise induced by powering-on/off PUs in different locations in MPSoC is evaluated. Assuming all the PUs induce the same supply current and have identical capacitance density, an inverter is put between a power grid node and its adjacent ground grid node to represent the PU switching activity. So there are as many inverters as the P/G nodes in our models. For example, for one PU, we use 43 560 inverters to mimic its behavior. The

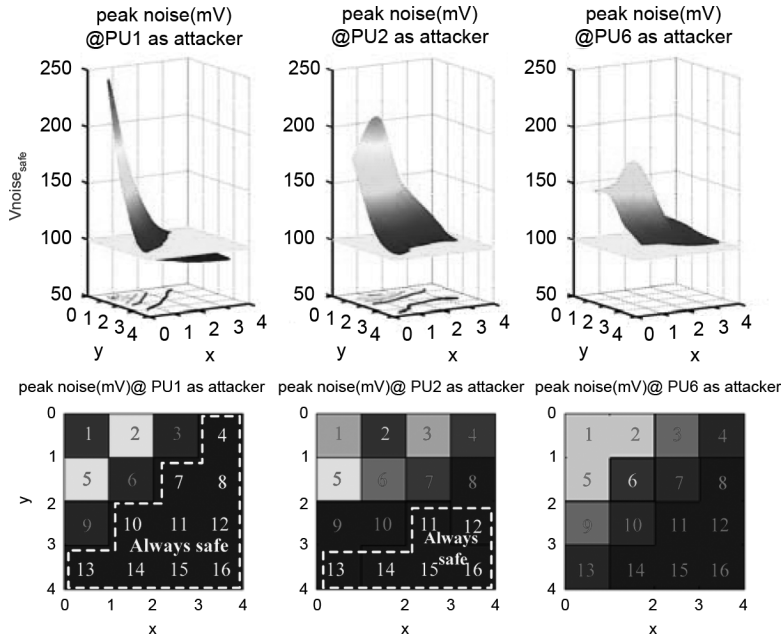

Fig. 8. Noise level and impact range of power gating induced P/G noise in 16-PU-MPSoC.

inverter size is chosen according to the average power consumption requirement for typical PUs. For different MPSoCs, the average power of each PU is about 30 mW, and the peak power of each PU is about 290 mW. We get these two values based on ARM11 [27] by scaling down the process parameters. For example, the worst-case noise by a PU switching can be simulated as all inverters of the switching PU charging and the others discharged, because the discharged inverters cannot help to reduce power/ground noise as decoupling capacitance. In the simulation, the PTM 45 nm bulk CMOS model [23] is used for transistors ($V_{dd} = 0.8$ V). The standard cell library is from the Nangate Open Cell Library [28]. The clock frequency is set to 1 GHz. We assume that 15% of the total chip area is occupied by decoupling capacitors.

1) P/G Noise Generation and Propagation: A powering-on/off PU is defined as an *attacker*. A PU, which carries an active task, is defined as an *active* PU. An active PU within the impact range of an attacker is defined as a *victim*. (Please note that some power-on PUs could be idle, and they are not victims in our definition.)

The temporal influence of P/G noise in MPSoC is illustrated in Fig. 6. We obtain the impact of P/G noise on standard logic cells and D flip-flop through simulation. Safe voltage levels are set for different cells to satisfy the performance requirement, and then a safe voltage level for a PU can be calculated as V_{safe}.

T^i_{settle} is the minimum time required for the voltage difference between V_{dd} and V_{ss} node pairs of an attacker i to be stabilized above the safe voltage level V_{safe}. The measurement of T^i_{settle} is started at the beginning of the switching event. T^{ij}_{safe} is the earliest time instance after which $V_{dd} - V_{ss}$ of a victim j is stabilized above V_{safe}. These parameters are extracted from our P/G network simulation.

2) Safe Voltage Level for PU (V_{safe}): We use a variable to get the relationship between delay variation and the V_{dd} variation. We use standard logic cells, including BUF, AND2, NAND2, OR2, NOR2, XOR2, etc., in the standard cell library. The noise voltage source plus the ideal V_{dd} connects between nodes V_{dd} and V_{ss} of each standard cell. The noise voltage source is a damped sinusoidal source that is the product of a dying exponential with a sine wave. The damped sinusoidal noise is analytical, and it can approximately imitate the P/G noise. Fig. 7 shows the transition delay variation of the standard cells under different P/G noise levels. Compared with ideal V_{dd}, in most cases, the supply voltage variation makes the delay increase (we just show the increased delay in Fig. 7); and in several cases, it makes the delay decrease. Increased delay will lead to performance degradation, while decreased delay may introduce racing risk. Hence P/G noise will threaten the performance or reliability of circuits. Proceeding from most cases shown in Fig. 7, if the delay relaxation is set to 20%, i.e., the circuit will be safe under 20% delay variation, the noise safe voltage should be 100 mV.

3) Example for MPSoC With 4×4 PUs: We illustrate the benefit of the noise aware processor level power gating strategy by taking a MPSoC with 4×4 PUs as an example.

First, we will introduce the conservative strategy—**stop-go** method. During powering on or off a PU, the stop-go strategy protects all the other active PUs. The stop-go algorithm is simpler and safe, but it is conservative according to our P/G noise model.

Second, the peak P/G noise levels of PUs induced by attackers located at different locations are shown in Fig. 8. Different impact areas can be observed: for PU1, at most 5 PUs need protection; for PU2, at most 9 PUs need protection; for PU4, all the other PUs need protection. $R_{\text{impact}}(p)$ is defined as the set of the victims of an attacker p.

Fig. 9 shows the voltage difference between V_{dd}/V_{ss} grid node pair when PU1 is the attacker. Peak noise of each PU is given in Table I. In Table I, D represents distance from the active PU (PU1) to the other PUs and the side length of a single PU is set to be 1. N represents peak noise of PUs.

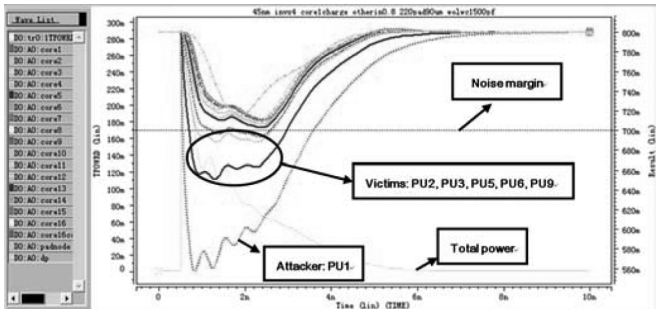

Fig. 9. Waveforms of voltage difference between vdd/vss grid node pair when PU1 is charging and others are already discharged.

To assign a new task, the conservative strategy—**stop-go** method will protect all the active PUs (clock gating them) when any PU is turning on. However, based on our P/G noise model, when the new task is assigned to different PUs, there exist some PUs which do not need protection, e.g., if the new task is assigned to PU1, 10 PUs within the *always safe* range do not require protection, and at most 5 PUs will need to be clock gated. Hence we can improve the MPSoC performance and reduce the noise protection penalty by carefully allocating the tasks onto PUs with different impact ranges.

IV. Power Gating Aware Task Assignment and Scheduling in MPSoC

A. Task and System Model

The task model is represented by a directed acyclic graph (DAG), in which nodes represent tasks and directed edges indicate data dependencies between tasks. Let Task be the set of tasks, and $t(i)$ be the execution time of task $i \in \text{Task}$.

A homogeneous MPSoC is a set of PUs, $PU = \{p | p \in [1, 2, \ldots, N-1, N]\}$, in which N is the total number of PUs. If we assign a task i to PU p, the powering-on/off noise when the task begins/finishes will attack the PUs in $R_{\text{impact}}(p)$, which is provided through our P/G model. The noise protection method is to clock gate the victim PUs and to wake up them when the attacker is fully turned on or off. $PA(p)$ is defined as the set of potential attackers of PU p, which can be easily derived from the impact ranges of the on-chip PUs.

When an attacker powers on/off, we define the noise protection penalty of a victim PU to be $TI_{\text{on}} = T_{\text{settleon}} + T_{\text{clkoff}} + T_{\text{clkon}}$ and $TI_{\text{off}} = T_{\text{settleoff}} + T_{\text{clkoff}} + T_{\text{clkon}}$, respectively, where T_{clkoff} and T_{clkon} are the time needed to clock gate a PU and wake it up from the clock gating state, and T_{settleon} and $T_{\text{settleoff}}$ are the settle time for a PU to power on/off.

B. MILP Formulation for Static P/G Noise Aware Power Gating Strategy

1) Problem Definition: Given a homogeneous MPSoC with N PUs, and a set of real-time tasks Task, determine a static assignment of tasks to PUs, such that all task constraints and deadlines are met, the MPSoC performance is optimized and the penalty for safeguarding the victim PUs is minimized.

We present the first MILP formulation for this problem. The objective function of our MILP has two parts: 1) the total execution time T_{end} for all the tasks and 2) the performance overhead for protecting the victims from the power gating induced P/G noise: $\sum_{p \in PU, i, j \in \text{Task}} (\alpha\text{spa}(i,j) \times TI_{\text{on}} + \alpha\text{fpa}(i,j) \times TI_{\text{off}})$.

The variables used in the MILP are defined in Table II. Several key constraints making our model different from other techniques are listed as follows.

1) Special timing constraints considering protection and settling times. Fig. 10 shows the extreme conditions of two tasks' start and finish time no matter to which PUs they are assigned. These extreme conditions help us to determine the timing constraints.
   For the first case in Fig. 10 1): task i starts (powering on a PU) right before task j's finish time (turning off a PU). $ts(i)$ and $tf(j)$ should satisfy the timing constraints: $\forall i, j \in \text{Task}, i \neq j : ts(i) \leq tf(j) - TI_L$ so that the victim protection procedure for these two attack operation will not conflict. TI_L are defined as: $TI_L = T_{\text{settleon}} + T_{\text{settleoff}} + T_{\text{clkoff}} + T_{\text{clkon}}$.
   Similarly, for the case in Fig. 10 2): task i finishes right before task j's finish time: $tf(i) \leq tf(j) - TI_{\text{off}}$; for Fig. 10 3): task i starts right before task j's start time: $ts(i) \leq ts(j) - TI_{\text{on}}$; for Fig. 10 4): task i finishes right before task j's start: $tf(i) \leq ts(j) - TI_S$. Here, $TI_S = T_{\text{clkoff}} + T_{\text{clkon}}$. These constraints are introduced in the MILP formulation to guarantee the tasks are assigned to PUs one by one without overlap.
2) The following constraints on $\alpha s(i,j)$ are used to model the case where task i starts during the execution of task j

$$\forall i, j \in \text{Task}, i \neq j, \forall p \in PU :$$
$$tf(j) - TI_L \geq ts(i) + (\alpha s(i,j) - 1) T_{\text{cons}} \quad (1)$$
$$ts(j) + TI_{\text{on}} \leq ts(i) + (1 - \alpha s(i,j)) T_{\text{cons}} \quad (2)$$
$$2 \geq \alpha s(i,j) + x(i,p) + x(j,p) \quad (3)$$
$$tf(j) + TI_S - p(i,j) \times T_{\text{cons}} \leq ts(i) + \alpha s(i,j) T_{\text{cons}} \quad (4)$$

   where the first two inequations are used to make sure the start time of task i is between the start and end time of task j; the third inequation is to make sure task i, j are not executed on the same PU, when $\alpha s(i,j)$ is true; the fourth is to make sure if i does not start during j and if j starts earlier than i, the starting time of task i should be larger than the finishing time of task j; T_{cons} is a constant larger than the worst case execution time to finish all the tasks. The timing constraints for $\alpha f(i,j)$ are similar and omitted here.
3) Execution time $t(i)$ and finishing time $tf(i)$ for each task i:

$$\forall i, j \in \text{Task}, \forall p, q \in PU :$$
$$t(i) = t_{\text{ori}}(i) + T_{\text{settleon}} + T_{\text{settleoff}} + \sum_{j \in \text{Task}} (\alpha\text{spa}(j,i) \times TI_{\text{on}} + \alpha\text{fpa}(j,i) \times TI_{\text{off}}) \quad (5)$$
$$t(i) + ts(i) \leq tf(i) \leq t(i) + ts(i) + \text{numTasks} \times TI_L \quad (6)$$

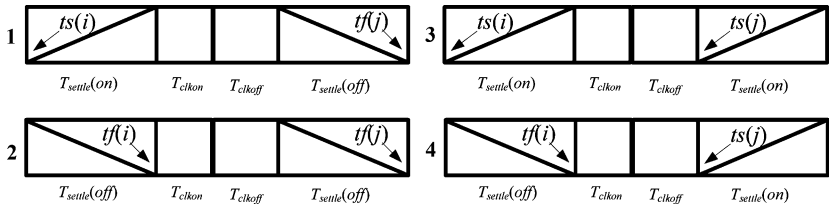

Fig. 10. Four conditions of two tasks' starting and finishing time.

TABLE II
VARIABLES USED IN THE MILP

$x(i,p)$	set of 0-1 variables s.t. $x(i,p)=1$ iff Task i is assigned to PU p
$ts(i)$	Execution start time for Task i
$tf(i)$	Execution finish time for Task i
$t(i)$	Execution time for Task i
T_{end}	Finish time for all the Tasks
$p(i,j)$	set of 0-1 variables, s.t. $p(i,j)=1$ iff $ts(i) \le ts(j)$
$e(i,j)$	set of 0-1 variables, s.t. $e(i,j)=1$ iff $tf(i) \ge tf(j)$
$\alpha s(i,j)$	set of 0-1 variables, s.t. $\alpha s(i,j)=1$ iff $ts(j) < ts(i) < tf(j)$
$\alpha f(i,j)$	set of 0-1 variables, s.t. $\alpha f(i,j)=1$ iff $ts(j) < tf(i) < tf(j)$
$\alpha spa(i,j)$	set of 0-1 variables s.t. $\alpha spa(i,j)=1$ iff assigning Task i ATTACKs Task j $=\alpha s(i,j) \times \sum_{p\in PUs}(x(j,p) \times \sum_{q\in PA(p)} x(i,q))$
$\alpha fpa(i,j)$	set of 0-1 variables s.t. $\alpha fpa(i,j)=1$ iff finishing Task i ATTACKs Task j $=\alpha f(i,j) \times \sum_{p\in PUs}(x(j,p) \times \sum_{q\in PA(p)} x(i,q))$

where $t_{\text{ori}}(i)$ is the predicted execution time of task i; $\text{numTasks} \times TI_L$ is to give each task a slack so that each task can be turned off one by one.

The left constraints, including precedence constraints of task pairs, task execution constraints, task deadline constraints, linearization constraints (for $\alpha\text{spa}(i,j)$ and $\alpha\text{fpa}(i,j)$) are as follows.

1) Every task i is assigned to exactly one PU

$$\forall i \in \text{Task} : \sum_{p\in PU} x(i,p) = 1. \tag{7}$$

2) Deadlines for each task i

$$\forall i \in \text{Task} : tf(i) \le D(i). \tag{8}$$

3) Precedence for each task pairs

$$\forall i,j \in \text{Task}, i \ne j :$$
$$p(i,j)+p(j,i)=1, e(i,j)+e(j,i)=1 \tag{9}$$
$$ts(i)+TI_{\text{on}} \le ts(j)+(1-p(i,j))T_{\text{cons}} \tag{10}$$
$$tf(i)+TI_{\text{off}} \le tf(j)+(1-e(i,j))T_{\text{cons}} \tag{11}$$
$$ts(i) \times \text{dep}(i,j) \ge tf(j)+TI_S \tag{12}$$

where $\text{dep}(i,j)$ is a Boolean constant that $\text{dep}(i,j)=1$ represents task i depends on task j in the task graph.

4) Extra Precedence constraints for tasks on the same cores:

$$\forall i,j \in \text{Task}, i \ne j, \forall p \in PU :$$
$$tf(i)+TI_S \le (2-x(i,p)-x(j,p))T_{\text{cons}}+ts(j) + (1-p(i,j))T_{\text{cons}} \tag{13}$$
$$tf(i)+TI_S \le (2-x(i,p)-x(j,p))T_{\text{cons}}+ts(j) + (1-e(i,j))T_{\text{cons}}. \tag{14}$$

SA Algorithm for P/G noise aware Task Scheduling

Input: the Task set *Task*, execution time of each task *t(i)*; the PU set *PU* and the impact relation between PUs.
Output: the assignment of the tasks to PUs and the end time.

```
1  Temp=Initial Temperature; Sol=Initial random solution;
2  MinResult=Evaluate(Sol); MinSol=Sol;
3  t=0; Iteration=0;
4  do
5        Sol'= Neighbor(Sol);
6        NewResult=Evaluate(Sol');
7        if NewResult<=MinResult
8              MinResult=NewResult; MinSol=Sol; Sol=Sol';
9        else if exp((MinResult-NewResult)/Temp)>Random(0,1)
10             Sol=Sol';
11       t++; Temp=A/(1+B*t); Iteration++;
12 while Temp>=1 && Iteration<MaxIteration;
```

Fig. 11. Simulated annealing algorithm for P/G noise aware task scheduling.

5) Linearization constraints for $\alpha\text{spa}(i,j)$ and $\alpha\text{fpa}(i,j)$ follows the linearization constraints for $C=B\times A$: if the upper bound of B is M, and A is Boolean variable, C can be relaxed as: $0 \le C \le B$; $C \le M \times A$; $C \ge B - M \times (1-A)$.

V. EFFICIENT ALGORITHMS FOR STATIC POWER GATING AWARE TASK SCHEDULING

The MILP formulation serves as a theoretical starting point of heuristic algorithms. However, it cannot be used to efficiently solve large problem instances, especially when the problem is NP-hard [29]. In this section, we propose a stochastic algorithm based on Simulated Annealing (SA) and an alternative Heuristic algorithm (HA) to address the problem of static task scheduling with power gating awareness.

A. SA Algorithm

Fig. 11 shows the pseudo-code for the simulated annealing algorithm. In the initialization, an initial solution is randomly generated (line 1) and evaluated (line 2), that is, each task is randomly assigned to a PU, and all the tasks on the same PUs are ordered by the topological order. Then, the temperature is set to an initial temperature and the algorithm begins annealing (line 3).

In each annealing cycle, a new neighbor solution is generated from the previous best solution. If the new solution is better than the previous one, we will accept it. But on the contrary, if the new solution is worse, we will accept it with the probability exp((MinResult-NewResult)/Temp), otherwise we drop the neighbor solution and start the next cycle from the best solution so far. The annealing process iterates along with the temperature decreases until the temperature becomes smaller than

Power Gating induced P/G noise aware Task Assigning and Scheduling

Input: the Task set *Task*, execution time of each task *t(i)*; the PU set *PU* and the impact relation between PUs.
Output: the task assignment and the end time.

```
1  Abstract the task set into a DAG;
2  Topological Ordering;
3  Calculate the theoretic static timing information of each task;
4  Decide the critical tasks;
5  Time node t=0;
6  do
7        update RS (Ready Set), CS (Critical Set);
8        if there exists a task T_i finish at time node t
9              power off the PU of T_i;
10       for each task in CS do
11             assign the task to an off PU which has minimum num of Victims;
12       end
13       for each non-critical task T_i in RS do
14             if require time of T_i is much bigger than t
15                   continue;
16             elseif require time of T_i is near t
17                   if there exists an off PU with minimum num of Victims
18                         assign T_i to this PU
19                   elseif T_i can be treated as critical task
20                         put T_i into CS
21       end
22       t++;
23 while there exists an un-finished task
```

Fig. 12. Fast heuristic algorithm for P/G noise aware task scheduling.

1 or the iteration count achieves the limit. The annealing cycle is shown from line 4 to 12.

The neighbor function (line5) plays a key role in the SA algorithm: given a valid assignment, another valid assignment needs to be generated by the neighbor function. In the neighbor function, first, a PU p is randomly chosen, one task from the task queue of PU p is selected, and then moved to another random PU q. Based on the task graph and the timing constraints, we choose the earliest start time for the moved task on PU q. This neighbor function ensures all the timing constraints are satisfied.

1) Complexity Analysis: For each cycle, we need to evaluate the current solution, the running time complexity of this step is $\mathrm{O}(N_{\mathrm{task}} \times T_{\mathrm{sum}})$ and the cycle count is maximum number of the temperature nodes N_{iter} which can be controlled by the user. So the worst case complexity of SA algorithm is $\mathrm{O}(N_{\mathrm{task}} \times T_{\mathrm{sum}} \times N_{\mathrm{iter}})$, where N_{task} is the number of tasks and T_{sum} is the sum of expected execution time of all tasks.

B. Heuristic Algorithm

A heuristic algorithm is further proposed for offline power gating aware task scheduling, as shown in Fig. 12.

In the initialization step (line1–5), we first abstract the task set into a Directed Acyclic Graph (DAG). Every node in the graph represents a task and a directed edge (i, j) represents task j depends on task i, which means task j must start after the finish time of task i. Then all the tasks in the DAG are ordered into a topological order. Based on the DAG and the topological order, we calculate the earliest start time of each task and the total execution time of all the tasks, further more the require time of each task can also be calculated. Thus, we can get the slack time of each task and decide the critical tasks whose slack time is 0.

In the task assignment loop (line6–23), we keep two sets ready set (RS) and critical set (CS). The RS consists of all tasks whose precedent tasks have already been finished. The CS contains the tasks whose slack time at time t is 0. We move the critical tasks from RS into CS. The procedure iterates as the time advanced until all the tasks are finished.

At the beginning of each iteration, if some tasks finish at this time instance, check the timing constraints and tentatively power off the PUs with the finished tasks (line 8). If a PU can not be powered off according to the timing constraints, it will be changed to the idle state until it can be powered off or another task is assigned to it. For each task in CS, if there exist some spare PUs satisfying the timing constraints, then we assign it to the PU who has minimum number of victim PUs (line 10–12). For each noncritical task in RS, if its require time is still far away from current time instance t, we will not assign it so that spare PUs are kept for possible upcoming critical tasks. If the require time of the noncritical task is *near* the current time instance, we investigate the PUs with minimum number of victims: if the victim number is small, then we assign the task, otherwise we don't. If the require time of a task in RS is approaching current time instance t, this task becomes a critical one, then it will be assigned following the rules of critical tasks. The complexity of HA is $O(N_{\mathrm{task}})$.

1) Complexity Analysis: For each cycle, we will consider all the tasks in CS and RS whether they should be assigned, and the cycle count is smaller than T_{sum}. So the worst case complexity of HA algorithm is $\mathrm{O}(N_{\mathrm{task}} \times T_{\mathrm{sum}})$.

VI. Lightweight On-Line Tasks Scheduling Methods

Since the actual execution time of a task at run-time may vary from its predicted value at design time, the static task scheduling may not be applicable for realtime implementation. Increased and decreased execution time of an attacker will cause potential reliability threaten to its victims. Here, a lightweight online adjustment strategy is provided to ensure the reliability of MPSoC.

A task execution time at runtime may vary from the predicted value in the following two ways—increasing or decreasing.

1) If the online monitors find the real execution time of task i on PU p to be longer than the predicted one, all the PUs on chip will be protected at the predicted power-off time of PU $(tf(i))$ until this PU is really powered off when task i is finished (the real finish time is denoted as $tf_{\mathrm{real}}(i)$). The original static task scheduling (task start time and protection time) for tasks after $tf(i)$ should be postponed by $tf_{\mathrm{real}}(i) - tf(i)$.
2) If the real execution time of task i on PU p is shorter than the predicted one, the online adjustment strategy conservatively keeps PU p in the idle state and power it off at the predicted power off time $tf(i)$. The original static task scheduling afterwards will not be influenced.

VII. Implementation and Simulation Results

A. Implementation and Experiment Setup

The P/G noise analysis platform is built up with Hspice and C. The MILP model is solved by SCIP [30]. SA and HA algorithms are implemented with C. The experiments are performed on a server with 2 Intel Core2 Xeon and 8 GB memory.

1) P/G Noise Parameters for MPSoC: The P/G network parameters are extracted from PTM interconnect model [23]. For different MPSoCs, the average power consumption of a single

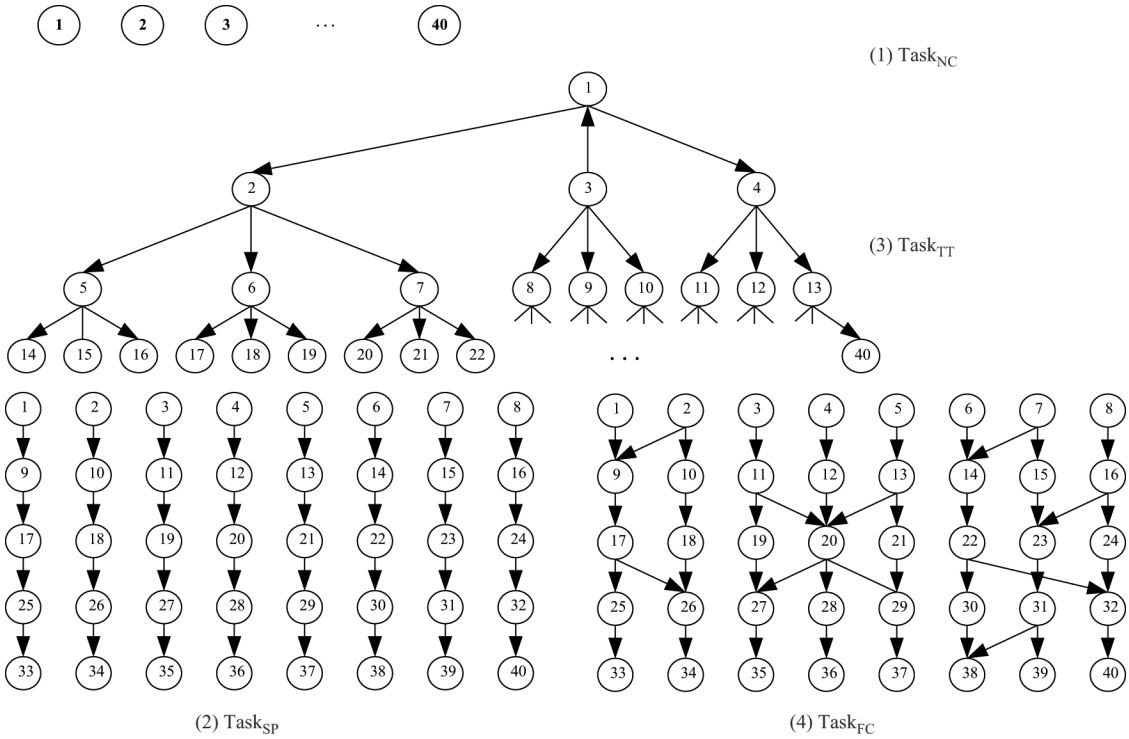

Fig. 13. Four task structures in our experiments: 1) TASK_{NC}: Tasks with No Correlation; 2) TASK_{SP}: several Sequential tasks in Parallel; 3) TASK_{TT}: Tree-connected Tasks; 4) TASK_{FC}: Fully Correlated tasks.

PU is kept as a constant around 30 mW, and the area of a single PU is set as a constant 660 μm$\times$660 μm, which also comes from ARM11 by scaling down the process parameters. Based on the performance results of standard logic cells and D flip-flops with different P/G noise, the noise toleration of $V_{dd}-V_{ss}$ is set as 100 mV, hence V_{safe} defined in Section II-B is set to 700 mV. Then the corresponding impact range $R_{\mathrm{impact}}(p)$ of each attacker p is derived for 4×4 to 8×8 PU mesh MPSoCs. T_{settle} for both powering on and off are set to 200 clock cycles. We choose this particular value based on [31]. As mentioned in [31], the power gating technique that reduces the wake-up time to shorter than 1 μs has been developed and the wake-up time of a 2 M gates circuit is around 300 ns under certain conditions. We can calculate that the scale of a single PU in our model is about 0.77 M gates. The frequency of our system is 1 GHz. So it is acceptable to choose 200 clock cycles as T_{settle}. The T_{clkon} and T_{clkoff} is set to be 100 clock cycles for the time penalty to protect/resume the data and clock-on/off the victim PU according to T_{settle}. All the time units in the results are measured by clock cycles.

2) Task Benchmarks for MPSoC: Four task structures are generated as test benchmarks: (1) TASK_{NC}: Tasks with No Correlation; 2) TASK_{SP}: several Sequential tasks in Parallel; 3) TASK_{TT}: Tree-connected Tasks; 4) TASK_{FC}: Fully Correlated tasks (a connected DAG with multiple inputs and multiple outputs).

In each task structure, we generate test cases with different number of tasks. We set the predicted execution time of a task randomly between 1 to 20 000 clock cycles for typical conditions. At the end of Section VII-B2) we will also discuss the conditions that the task length becomes longer: up to 200 K clock cycles. Fig. 13 illustrates the examples of the four types of task structures with the number of tasks equals to 40.

TABLE III
MILP AND SA FOR TASK_{NC} ON 4×4 MPSoC. RUNTIME OF MILP IS LIMITED TO 10 hr, WHILE RUNTIME OF SA IS LIMITED TO 20 min

Task #	MILP		SA		Stop-go	
	T_{end}	CP/PT	T_{end}	CP/PT	T_{end}	CP/PT
6	20040	2/12	20040	2/12	23110	30/12
8	15340	6/16	15340	16/16	20940	56/16
10	20080	32/20	19610	30/18	26810	90/20
12	20720	50/24	19860	44/22	27060	132/24

B. Results For Power Gating Aware Task Scheduling

1) Comparison of MILP, SA, and Stop-Go On Task Sets With Small Number of Tasks: According to the four task structures, TASK_{NC} is the hardest scheduling benchmark with the largest solution space. We first evaluate some simple examples to compare the MILP/SA/stop-go methods shown in Table III. **CP** and **PT** denote the number of clock gating and power on/off operations, respectively. Both MILP and SA method achieve impressive improvement on end-to-end delay T_{end} compared with the stop-go method: from 10% to 30% and this improvement will be greater if the task number becomes larger.

The MILP and SA approaches obtain nearly the same results for six and eight tasks. However, for 10 and 12 tasks, SA method gets better results because in SA and HA algorithms, new tasks are allowed to run continuously on a PU if the previous task is just finished. While in MILP solution, in consideration of the complexity, our MILP model assumes that: a PU should be powering off when it finishes a task and can't receive a new one until it is powering on again. Hence, powering-on/off times (PT) of SA is smaller than PT of MILP. We tried task number larger than 12, the MILP model needs to run hours to get a sub-optimal result, hence only SA and HA are evaluated for larger problem instances.

TABLE IV
SA/HA/STOP-GO METHODS ON 40 TASK SCHEDULING ON4 × 4 MPSOC

Task Structure	Task length	SA			HA			Stop-go		
		T_{end}	CP/PT	Runtime	T_{end}	CP/PT	Runtime	T_{end}	CP/PT	Runtime
NC	variable	39850	200/42	30min	49380	326/50	3s	65940	960/80	4s
	equal	42500	178/38	21min	45100	316/52	3s	61200	536/80	4s
SP	variable	74920	104/60	45min	79620	116/66	4s	101240	434/80	9s
	equal	63130	90/42	37min	65200	126/60	4s	80800	280/80	6s
TT	variable	66140	160/54	45min	70630	270/62	5s	88470	652/80	7s
	equal	62210	122/38	40min	63500	204/44	4s	80800	428/80	6s
FC	variable	77980	66/60	50min	81040	88/62	5s	104380	368/80	9s
	equal	61400	68/38	41min	65000	106/54	5s	80800	280/80	6s

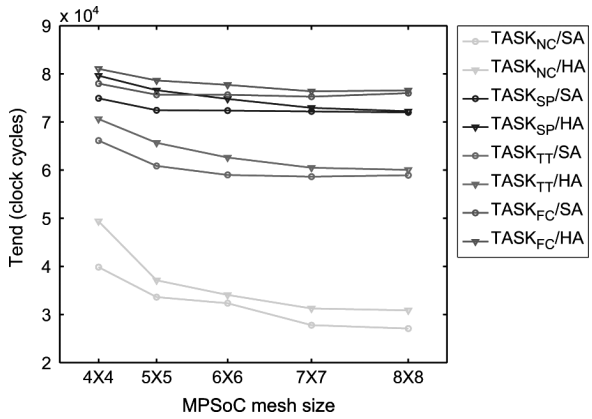

Fig. 14. Four kinds of tasks with fixed task number on different MPSoCs.

2) SA/HA/Stop-Go Results Comparison: Table IV shows SA/HA/stop-go results for 40 tasks of 4 task structures on 4×4 MPSoC. Both SA and HA methods achieve impressive $T_{\rm end}$ improvement compared with the stop-go method: from 21.9% to 39.6% and from 19.3% to 26.4%, respectively. CP of SA is up to 70% less than that of stop-go method. Meanwhile, SA gets better results than HA. PT and CP of SA are about 17.3% and 36.3% less than those of HA, respectively. And $T_{\rm end}$ of SA is about 6% better than that of HA. However, SA needs to run nearly an hour, while HA needs only several seconds. So designers can choose different algorithms to tradeoff between MPSoC performance and algorithm running time. We also use tasks with identical (equal length) and different execution times (variable lengths). For all the task structures, equal-length tasks will leads to smaller CP than variable-length tasks, because tasks with different lengths have higher possibility to intersect on the time domain.

In Fig. 14, more PUs will lead to smaller $T_{\rm end}$ but the trends are different for different task structures. For TASK_{FC} and TASK_{SP}, in which tasks are highly correlated, the improvement is limited. For TASK_{NC} with a larger solution space, $T_{\rm end}$ on 16-PU-MPSoC to 49-PU-MPSoC decreases quickly, but T_{end} on 64-PU-MPSoC is close to that on 49-PU-MPSoC. This is because the task number is not large enough to exert the advantage of a larger PU number. For TASK_{TT}, the decreasing rate is in the middle, since in the root level there are few parallel tasks, when the tree becomes deeper, there will be more parallel tasks to fully utilize the increased PU number. Fig. 14 also shows that $T_{\rm end}$ improvement of SA is larger than that of HA, however this difference will be smaller if the PU number becomes larger except TASK_{NC}.

Table VI shows the results for TASK_{NC} and TASK_{FC} with different task numbers on 16/64-PU-MPSoC. In this table, more PUs lead to smaller $T_{\rm end}$, especially when the task number becomes larger. For $rmTASK_{NC}$ with larger solution space, $T_{\rm end}$ on 64-PU-MPSoC is less than half of that on 16-PU-MPSoC. For TASK_{FC} in which tasks are highly correlated, the improvement is very limited even when the task number increases. CP decreases with increased PU number, since there are more safe assignment choices: more PUs are not attacked by the powering-on/off of a single PU. On the contrary, CP for stop-go method will increase since the influenced PU number increases with increasing total PU number.

For longer tasks up to 200 K cycles, Table V shows SA/HA/stop-go results for 40 tasks of NC task structure on 4×4 MPSoC. In Table V, as the execution time of a single task becomes longer, the $T_{\rm end}$ improvement of both SA and HA methods compared with the stop-go method goes lower: from 28.6% to 2.04% and from 22.8% to 1.32%, respectively. That is because the efficiency of SA and HA methods is highly related to the number of tasks. Large number of tasks will highlight the impact of SA and HA methods. However, the number of tasks is set to be a constant 40 here. Meanwhile the clock gating penalty, about 200 cycles, will be negligible comparing with the ever-growing task length. Hence, when the execution time of a single task becomes long, the efficiency of SA and HA methods compared with the stop-go methods will become relatively low. On the other hand, CP of both SA and HA is at least 35.2% less than that of stop-go method no matter what the task length range is. And SA method shows smaller number of CP comparing with HA method, while HA method is much faster than SA method. Therefore, when the task length becomes longer, our methods will still effectively reduce the protection penalty when power gating technique is used in the low power MPSoC.

3) Tradeoff Between Clock Gating Penalty and End Time $T_{\rm end}$: Fig. 15 shows CP with $T_{\rm end}$ relaxation from 0% to 20%. We use TASK_{NC} and TASK_{FC} with 80 tasks as test benchmarks. The task number 80 is 5 times as the PU number of 4×4 MPSoC, hence there will be more timing slack for each task when $T_{\rm end}$ relaxation becomes larger, hence the CP on 4×4 MPSoC decreases continuously with both two task structures. Up to 40% CP is reduced when $T_{\rm end}$ relaxes 20%. However, for 8×8 MPSoC, CP decreases less obviously since the task number is similar to the number of PUs.

TABLE V
SIMULATION RESULTS OF SA/HA/STOP-GO METHODS ON 40 Task_{NC} SCHEDULING ON4 × 4 MPSoC

Task Length	SA			HA			Stop-go		
	T_{end}	CP/PT	Runtime	T_{end}	CP/PT	Runtime	T_{end}	CP/PT	Runtime
0-20000	46960	265/54	5min	50750	394/60	0.06s	65770	960/80	0.1s
40000-60000	1387480	564/78	3h54min	1448000	596/78	1s	147960	960/80	1s
80000-100000	2620460	559/76	5h2min	2699080	626/80	2s	2700850	960/80	2s
120000-160000	4056830	608/78	8h11min	4123900	622/80	4s	4123610	960/80	4s
180000-200000	5607690	561/76	29h56min	5648860	572/76	5s	5724650	960/80	5s

TABLE VI
HA WITH DIFFERENT TASK NUMBERS ON 16/64-PU-MPSoC

Task Bench + PU #	40 tasks		60 tasks		80 tasks	
	T_{end}	CP/PT	T_{end}	CP/PT	T_{end}	CP/PT
NC+16	49380	326/50	71440	660/88	85290	814/106
FC+16	81040	88/62	129810	168/104	156910	250/146
NC+64	30870	100/60	37970	240/84	41400	370/104
FC+64	76310	0/62	120180	0/100	145420	0/136

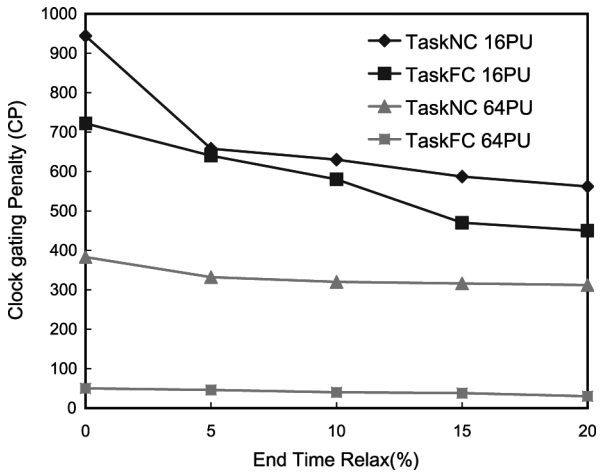

Fig. 15. CP versus T_{end} relaxation on both 4 × 4 and 8 × 8 MPSoC.

C. Online-Adjustment(OLA)

Since the worst-case execution time is normally used in static scheduling, the variation between real execution time and the predicted execution time is assumed to be -20% to 10%. We randomly generate the execution time for each task in TASK_{NC}. Since the OLA method ignores the variation of decreased execution time, T_{end} after OLA equals to the result of SA plus sum of increased execution time. The conservative stop-go method is easy to be carried out as another simple online strategy. Table VII shows the comparison of SA+OLA and online stop-go, our SA+OLA still leads to up to 30% T_{end} improvement. The upper bound of CP for SA+OLA can be estimated as: $CP_{SA} + N_{\mathrm{task+}} \times N_{PU}$. If the number of tasks with increased execution time $(N_{\mathrm{task+}})$ is small, we will still achieve less CP: for example, CP upper bound of 80 tasks on 16-PU-MPSoC using SA+OLA is $696 + 28 \times 16 = 1144$, about 50% of CP by online stop-go method: 2160.

VIII. CONCLUSION AND FUTURE WORK

In this paper, the power gating aware task scheduling problem based on our P/G noise analysis platform is formulated for MPSoC. Efficient methods, such as SA and HA are proposed for static scheduling at design time. The algorithms can improve the MPSoC performance while reduce the noise protection penalty. Extensive performance evaluation results show that the new methods can achieve considerable improvement compared with the conservative stop-go method. Furthermore, a light-weight online adjustment strategy is proposed together with the offline scheduling results to improve the chip reliability.

TABLE VII
STATIC SCHEDULING+ONLINE-ADJUSTMENT(SA+OLA) VERSUS ONLINE STOP-GO WITH TASK_{NC} ON4 × 4 MPSoC

Task #	SA SA	Decreased $t(i)$	Increased $t(i)$	SA+ OLA	T_{end} impr	On-line stop-go
20	21730	-12179	1042	22772	31.2%	33110
40	39850	-17776	8208	48058	28.8%	67484
60	59440	-33391	11788	71228	25.8%	96088
80	81210	-52538	16924	98134	16.2%	116852

For future work, the improvement can be done by 1) considering both the run time P/G noise induced by working current and the powering-on/off induced P/G noise and 2) combining P/G noise with thermal/power effects.

REFERENCES

[1] F. Mohamood, M. Healy, S. K. Lim, and H.-H. Lee, "Noise-direct: A technique for power supply noise aware floorplanning using microarchitecture profiling," in *Proc. ASP-DAC*, Jan. 2007, pp. 786–791.
[2] M. Healy, F. Mohamood, H.-H. Lee, and S. K. Lim, "A unified methodology for power supply noise reduction in modern microarchitecture design," in *Proc. ASP-DAC*, Mar. 2008, pp. 611–616.
[3] K. Shi and D. Howard, "Challenges in sleep transistor design and implementation in low-power designs," in *Proc. DAC*, Jul. 2006, pp. 113–116.
[4] S. Kim, S. Kosonocky, and D. Knebel, "Understanding and minimizing ground bounce during mode transition of power gating structures," in *Proc. ISLPED*, Aug. 2003, pp. 22–25.
[5] S. Kim, S. Kosonocky, D. Knebel, K. Stawiasz, D. Heidel, and M. Immediato, "Minimizing inductive noise in system-on-a-chip with multiple power gating structures," in *Proc. ESSCIRC*, Sep. 2003, pp. 635–638.
[6] K. He, R. Luo, and Y. Wang, "A power gating scheme for ground bounce reduction during mode transition," in *Proc. ICCD*, Oct. 2007, pp. 388–394.
[7] J. Gu, H. Eom, and C. Kim, "A switched decoupling capacitor circuit for on-chip supply resonance damping," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2007, pp. 126–127.
[8] H. Jiang and M. Marek-Sadowska, "Power-gating aware floorplanning," in *Proc. ISQED*, Mar. 2007, pp. 853–860.
[9] A. Davoodi and A. Srivastava, "Wake-up protocols for controlling current surges in MTCMOS-based technology," in *Proc. ASP-DAC*, Jan. 2005, pp. 868–871.
[10] A. Ramalingam, A. Devgan, and D. Z. Pan, "Walk-up scheduling in mtcoms circuits using successive relaxation to minimize ground bounce," *J. Low Power Electron.*, vol. 3, no. 1, pp. 1–8, 2007.
[11] H. Jiang and M. Marek-Sadowska, "Power gating scheduling for power/ground noise reduction," in *Proc. DAC*, Jun. 2008, pp. 980–985.
[12] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. DAC*, 2002, pp. 183–188.
[13] J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-aware scheduling under timing constraints for mission-critical embedded systems," in *Proc. DAC*, 2001, pp. 840–845.
[14] J. Hu and R. Marculescu, "Energy-aware communication and task scheduling for network-on-chip architectures under real-time constraints," in *Proc. DATE*, Feb. 2004, pp. 234–239.

[15] P. Rong and M. Pedram, "Power-aware scheduling and dynamic voltage setting for tasks running on a hard real-time system," in *Proc. ASP-DAC*, Jan. 2006, p. 6.
[16] A. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. DATE*, Apr. 2007, pp. 1–6.
[17] A. Coskun, T. Rosing, K. Whisnant, and K. Gross, "Temperature-aware MPSoC scheduling for reducing hot spots and gradients," in *Proc. ASPDAC*, Mar. 2008, pp. 49–54.
[18] T. Chantem, R. Dick, and X. Hu, "Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs," in *Proc. DATE*, Mar. 2008, pp. 288–293.
[19] A. Todri, M. Marek-Sadowska, and J. Kozhaya, "Power supply noise aware workload assignment for multi-core systems," in *Proc. ICCAD*, Nov. 2008, pp. 330–337.
[20] P. Gupta and A. Kahng, "Efficient design and analysis of robust power distribution meshes," in *Proc. 19th Int. Conf. VLSI Des. held jointly with 5th Int. Conf. Embed. Syst. Des.*, 2006, p. 342.
[21] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu, and D. Bearden, "Design and analysis of power distribution networks in powerpc microprocessors," in *Proc. 35th Annu. Conf. Des. Autom. ACM*, New York, 1998, pp. 738–743.
[22] K. Shakeri and J. Meindl, "Compact physical IR-drop models for chip/package co-design of gigascale integration (GSI)," *IEEE Trans. Electron Devices*, vol. 52, no. 6, pp. 1087–1096, Jun. 2005.
[23] Nanoscale Integr. Model. (NIMO) Group, Arizona State Univ., Tempe, "Predictive technology model (PTM)," 2007. [Online]. Available: http://www.eas.asu.edu/\~{}ptm/
[24] S. Pant and E. Chiprout, "Power grid physics and implications for CAD," in *Proc. DAC*, 2006, pp. 199–204.
[25] G. Huang, D. Sekar, A. Naeemi, K. Shakeri, and J. Meindl, "Compact physical models for power supply noise and chip/package co-design of gigascale integration," in *Proc. Electron. Components Technol. Conf.*, 2007, pp. 1659–1666.
[26] ITRS 2007. [Online]. Available: http://www.itrs.net/
[27] D. Cormie and A. Ltd, "The ARM11 microarchitecture," White Paper, 2002.
[28] Nangate, USA, "Nangate open cell library," 2008. [Online]. Available: http://www.opencelllibrary.org
[29] S. Leonardi and D. Raz, "Approximating total flow time on parallel machines," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, El Paso, TX, May 1997, pp. 110–119.
[30] Div. Scient. Comput., Dept. Opt., Konrad-Zuse-Zentrum für Informationstechnik, Berlin, Germany, "SCIP: Solving constraint integer programs," 2008. [Online]. Available: http://scip.zib.de/
[31] K. Kawasaki, T. Shiota, K. Nakayama, and A. Inoue, "A sub-μs wake-up time power gating technique with bypass power line for rush current support," in *Proc. IEEE Symp. VLSI Circuits*, 2008, pp. 146–147.

Yu Wang (S'05–M'07) received the B.S. and Ph.D. degree with honors from NICS Group, Electronics Engineering Department, Tsinghua University, Beijing, China, in 2002 and 2007, respectively, under the supervision of Prof. H. Yang (Tsinghua University) and Prof. Y. Xie (Pennsylvania State University).

He is currently an Assistant Professor with the Department of Electrical Engineering, Tsinghua University. His research mainly focuses on fast circuit analysis, low power circuit design methodology, reliability-aware circuit design methodology, application specific FPGA design, and on-chip communication strategies for MPSOC. He has authored and coauthored over 50 technical papers in refereed journals and conferences.

Dr. Wang has been a technical program committee member for a number of conferences including ICCAD, ASPDAC, ISLPED, ISQED, ISVLSI, ICFPT, etc.

Jiang Xu (M'02) received the Ph.D. degree from Princeton University, Princeton, NJ, and the B.S. and M.S. degrees from Harbin Institute of Technology China, Harbin, China, all in electrical engineering.

He was a Research Associate with Bell Labs from 2001 to 2002, and was a Research Associate of NEC Laboratories America from 2003 to 2005. Before joining Hong Kong University of Science and Technology as an Assistant Professor in 2007, he was with a startup company to develop ultra-low power multiprocessor for mobile platforms. He has published over 30 papers and book chapters. His research interests include network-on-chip, multiprocessor system-on-chip, computer architecture, low-power VLSI design, and HW/SW codesign

Yan Xu received the B.S. and M.S. degrees from NICS Group, Electronic Engineering Department, Tsinghua University, Beijing, China, in 2006 and 2009, respectively, under the supervision of Prof. H. Yang and Dr. Y. Wang.

She is currently an Engineer with the Design Technology Group, MediaTek Inc., Beijing, China. Her research mainly focuses on power/ground noise analysis, MPSoC scheduling, and electronic system level (ESL) design.

Weichen Liu (S'07) received the B.S. and M.S. degrees from the Department of Computer Science and Engineering, Harbin Institute of Technology, Harbin, China, in 2004 and 2006, respectively. He is currently pursuing the Ph.D. degree from the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, China.

His research interests include real-time scheduling and multiprocessor systems.

Huazhong Yang (M'97–SM'00) was born in Ziyang, Sichuan Province, P. R. China, on Aug. 18, 1967. He received the B.S. degree in microelectronic, and the M.S. and Ph.D. degree in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993, and 1998, respectively.

In 1993, he joined the Department of Electronic Engineering, Tsinghua University, where he has been a Full Professor since 1998. His research interests include chip design for communication and multimedia applications, synthesis of analog integrated circuits (IC), power estimation and synthesis of digital ICs, noise and delay estimation of deep submicrometer ICs, yield enhancement, optimization and modeling. He has been in charge of over 20 projects, including projects sponsored by the 863 program, NSFC, the 9th five-year national program and several international cooperation projects. He has authored and coauthored over 100 technical papers and 6 books.

Dr. Yang was recognized as 2000 National Palmary Young Researcher by NSFC.