A Case Study of On-Chip Sensor Network in Multiprocessor System-on-Chip

Yu Wang[1,2], Jiang Xu[2], Shengxi Huang[1], Weichen Liu[2], Huazhong Yang[1]
[1]EE. Dept, TNList, Tsinghua University, Beijing, China,
[2]Mobile Computing System Lab, Dept. of ECE
Hong Kong University of Science and Technology, Hong Kong, China

ABSTRACT
Reducing feature sizes and power supply voltage allows integrating more processing units (PUs) on multiprocessor system-on-chip (MPSoC) to satisfy the increasing demands of applications. However, it also makes MPSoC more susceptible to various reliability threats, such as high temperature and power/ground (P/G) noise. As the scale and complexity of MPSoC continuously increase, monitoring and mitigating reliability threats at run time could offer better performance, scalability, and flexibility for MPSoC designs. In this paper, we propose a systematic approach, on-chip sensor network (SENoC), to collaboratively detect, report, and alleviate run-time threats in MPSoC. SENoC not only detects reliability threats and shares related information among PUs, but also plans and coordinates the reactions of related PUs in MPSoC. SENoC is used and explained in our case study to alleviate the impacts of simultaneous switching noise in MPSoC's P/G network during power gating. Based on the detailed noise behaviors under different scenarios derived by our circuit-level MPSoC P/G noise simulation and analysis platform, simulation results show that SENoC helps to achieve on average 26.12% performance improvement compared with the traditional stop-go method with 1.4% area overhead in an 8*8-core MPSoC in 45nm.

Categories and Subject Descriptors
B.8 **[Performance and Reliability]**: Performance Analysis and Design Aids, C.0 [Computer Systems Organization: General]: Modeling of computer architecture.

General Terms
Design, Reliability.

Keywords
Sensor network, reliability, dynamic control, low-power, noise, power grid, system-on-chip

1. INTRODUCTION
Multiprocessor system-on-chip (MPSoC) is becoming a favorite choice to satisfy the ever-growing performance demanded by applications [1,2,3]. On one hand, shrinking feature size allows for more and better functions on MPSoC. On the other hand, it also makes MPSoC more susceptible to various reliability threats, such as high temperature and power/ground (P/G) noise. Improving reliability has become an important aspect of MPSoC design.

Tiny on-chip sensors can measure various run-time parameters, such as noise, temperature, switching activity, clock duty-cycle, and technology parameters [4,5,6,7,8]. Petrescu et al. proposed a signal integrity architecture to monitor various on-chip physical parameters, especially voltage and temperature [4]. Poirier et al. and McGowen et al. described the control system on a 90-nm Itanium processor which utilizes on-chip sensors to measure power and temperature and modulates voltage and frequency to optimize performance [5, 6]. Sohn et al. proposed a sensor-based solution for SRAM to overcome the uncertainty and fluctuation of device parameters [7, 8]. These works show that the information gathered by on-chip sensors can be used to effectively improve the reliability and performance of different functional units. Several methods are proposed to collect useful information at run time. Chan et al. proposed to use system management bus to communicate between IP cores and the thermal-aware power management IP for system-level power management [9]. Yin et al. proposed a hierarchical architecture to collect run-time parameters using network-on-chip (NoC) [10]. Ciordas et al. proposed a monitoring service framework to support the run-time observability of NoC behaviors and application debugging [11, 12].

As the scale and complexity of MPSoC continuously increase, a systematic approach that not only detects reliability threats but also mitigates such threats accordingly at run time could potentially offer better performance, scalability, and flexibility for MPSoC designs. In this paper, we propose a systematic approach, on-chip sensor network (SENoC), to collaboratively detect, report, and alleviate run-time threats in MPSoC. SENoC not only detects reliability threats and shares related information among processing units (PUs), but also plans and coordinates the reactions of related PUs in MPSoC. SENoC is integrated with NoC to ensure that critical information and decision is delivered in a timely fashion.

To highlight the details of our idea, SENoC is used and explained in a case study to alleviate the impacts of simultaneous switching noise in MPSoC's P/G network during power gating. Tight low power requirements force MPSoC to aggressively adopt low power techniques. While power gating can dramatically reduce leakage power in MPSoC, it exacerbates simultaneous switching noise on the power delivery network, and can result in performance degradation and even functional errors. SENoC offers an effective solution to simultaneous switching noise in

CASES'09, October 11–16, 2009, Grenoble, France.

MPSoC's P/G network. Based on the detailed noise behaviors under different scenarios derived by our circuit-level MPSoC P/G noise simulation and analysis platform, simulation results show that SENoC helps to achieve on average 26.198% performance improvement compared with the traditional stop-go method with 1.4% area overhead in an 8*8-core MPSoC in 45nm.

The rest of the paper is organized as follows. Section 2 gives an overview of SENoC. Serving as the basis of our case study, the circuit-level modeling for power gating induced P/G noise is presented in Section 3. In section 4, we focus on the behavior models of the key components in SENoC and formalize their mechanisms. Section 5 shows the implementation and simulation results. Section 6 concludes the paper.

2. AN OVERVIEW OF SENOC

Besides detecting run-time threats and sharing related information among PUs, SENoC also decides and coordinates the reactions of all the PUs in MPSoC. SENoC is composed of on-chip sensors, node agents, and a task manager (TM). It is integrated with network-on-chip (NoC) and uses NoC as the communication medium. Fig. 1 illustrates the SENoC for a 4*4-core MPSoC with a mesh-based NoC.

Figure 1. SENoC on a 4*4-core MPSoC

Tiny sensors are embedded inside and between PUs to measure various parameters, such as voltage and temperature. Under normal operating conditions, the parameters are within a safe range, and sensors only report them upon requests from the node agents. However, sensors will immediately report any parameter beyond the safe range to node agents. They are usually placed close to the functional units which have high power consumption or are sensitive to temperature or voltage conditions.

Node agents are integrated with NoC routers and connected to sensors through network interfaces. Each node agent processes the warnings from a group of sensors inside a PU. It will preprocess and filter warning as well as compress warnings from multiple sensors. Based on the warning severity, a node agent can send alert packets to nearby node agents or all the node agents in addition to TM. Node agents not only take commands for TM, but also collaborate with each other to quickly respond to a threat.

SENoC uses NoC as the communication medium. We integrate the node agent into an input-buffered pipelined NoC router (Fig. 2). The integration assures SENoC that node agents can send and receive packets in NoC in a timely fashion, which is critical to SENoC. NoC routers use virtual channels to guarantee the delivery time of high-priority alert packets. In addition to unicast, multicast is used to send packets simultaneously to multiple node agents. SENoC uses four packet types -- command packet, alert packet, report packet, and post-alert packet. TM processes packets and creates a system-wide plan to minimize threat impacts. It is connected to NoC and sends command packets to node agents to execute the plan. When receiving a command packet, node agents will inform the PU to take specific actions, such as entering sleep mode through clock gating.

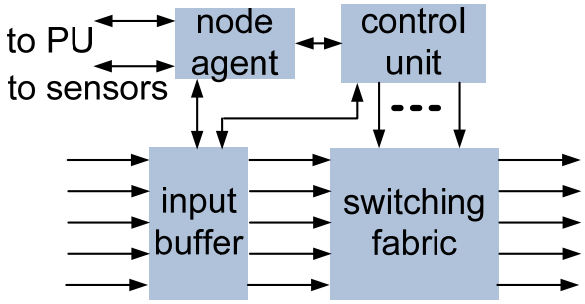

Figure 2. Integrated node agent in NoC router

3. THE PHYSICAL MODEL OF POWER GATING INDUCED P/G NOISE

SENoC is used in our case study to alleviate the impacts of simultaneous switching noise in MPSoC's P/G network during power gating. In order to analyze the performance of SENoC in the case study, it is necessary to understand the impacts of power gating induced P/G noise among PUs in MPSoC. We build a P/G noise simulation and analysis platform and systematically explore MPSoC P/G noise behaviors under different power gating scenarios [13]. The noise behaviors serve as the basis for the P/G noise aware task management methodology based on SENoC.

The simultaneous switching noise induced by turning on/off PUs at different locations in MPSoC is evaluated based on circuit-level simulations. Assuming all the PUs induce the same rush current and have identical capacitance density, inverters are placed between power grid nodes and adjacent ground grid nodes to represent PU switching activity. A small part of the P/G network circuit model is shown in Fig. 3. For the case study, we use the 45nm bulk CMOS model [14] for transistors (V_{dd}=0.8V) and the standard cell library is from the Nangate Open Cell Library [15] for the whole design.

MPSoC is modeled with a set of PUs, PU=PU_p; p=1, 2, …, N. The PU states considering power gating induced P/G noise are defined in Table 1. A ToOn/ToOff 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 or Free, and they are not victims in our definition.) For a PU *p*, R_{impact}^{p} is defined as the set of the victims of an attacker p, while *PV(p)* is defined as a set of PU *p*'s potential victims, namely *PV(p)={q| q* ∈

PU, q≠p and q is in the impact range of p}. The number of PU p*'s* potential victims is denoted by $N_{PV}(p)$.

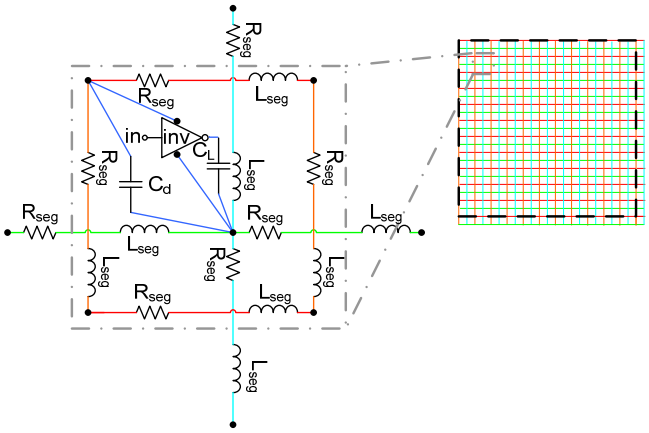

Figure 3. In order to facilitate the P/G network analysis, each wire segment is modeled as a chain of L-type RLC equivalent circuits. An inverter with a capacitance load is used to imitate logics. A decap is connected to the intersection points on the vdd/vss grids.

Table 1. PU states in power gating

PU States	Illustrations
Off state	Power gated.
ToOn state	The off to on transition. The start time of the transition for PU *p* to execute task *i* is defined as *ton(i)*.
ToOff state	The on to off transition. The end time of the transition for PU *p* just finished task *i* is defined as *toff(i)*.
Idle state	Clock gated (power is on).
ClkToOff state	Clock gate transition.
ClkToOn state	Clock wake up transition.
Free state	Both power and clock are on, but there is no task running on the PU.
Active state	A task is running on the PU.

If we assign a task i to a power gated PU *p*, the powering-on/off noise when the task begins/finishes will attack the PUs in R_{impact}^{p} which is provided through our P/G model. The noise protection method will migrate the data off chip, clock gate the victim PUs before powering on or off the attacker, and later wake up them when the attacker is fully turned on or off. Fig. 4 shows the timing of a power on event, and the timing of a power off event is similar. $T_{ClkToOff}$ and $T_{ClkToOn}$ are the time needed to clock gate a PU and to wake it up from the clock gated state, respectively. $T_{onsettle}$ *and* $T_{offsettle}$ are the settle time for a PU to power on and power off. In order to ensure the reliability of MPSoC, here $T_{onsettle} \geqslant \max\{T_{settle}^{p}, TI_{safe}^{pq}, p, q \in \text{PU}, q \neq p\}$, $T_{offsettle} \geqslant \max\{T_{settle}^{p}, TI_{safe}^{pq}, p, q \in \text{PU}, q \neq p\}$, where T_{settle}^{p} is the period that p powers on, and TI_{safe}^{pq} is the period that both p and q return to normal voltage level. TI_{ON} and TI_{OFF} are the noise protection time penalty for a victim PU when an attacker powers on and off respectively, where $TI_{ON} = T_{ClkToOff} + T_{onsettle} + T_{ClkToOn}$, $TI_{OFF} = T_{ClkToOff} + T_{offsettle} + T_{ClkToOn}$. Assume that the victim number of PU p as an attacker at the moment is $N_{victim}(p,t)$. we define $P_{on}(p,t)$ and $P_{off}(p,t)$ as the total performance penalty to power on and power off attacker p, respectively, where $P_{on}(p,t) = TI_{ON} \times N_{victim}(p,t)$, $P_{off}(p,t) = TI_{OFF} \times N_{victim}(p,t)$. Initially, these timing parameters, such as $T_{ClkToOff}$, $T_{onsettle}$, $T_{ClkToOn}$, $T_{offsettle}$ is the worst case value to ensure the reliability, however, with the help of SENoC, these parameter will be dynamically decided using on-chip sensors.

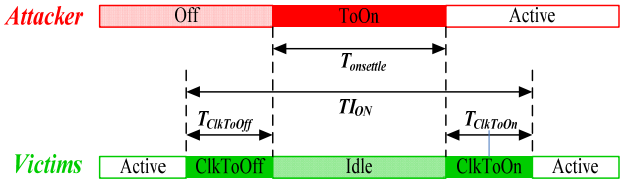

Figure 4. Timing of a power-on event.

Based on the P/G noise simulation and analysis platform, we simulate MPSoC with different scales and under different conditions. Fig. 5 shows the peak P/G noise levels of PUs induced by attackers located at different locations on a 4*4-core MPSoC. Different impact ranges can be observed: for PU1 as an attacker, at most 5 PUs need protection; for PU2, at most 9 PUs need protection; for PU6, all the other active PUs need protection. SENoC can help with the voltage level collection, noise information transfer, analysis, and planning. The detailed procedure will be shown in the next section.

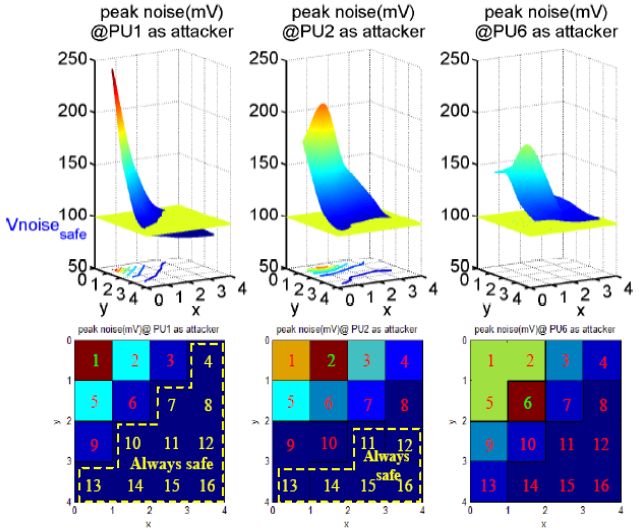

Figure 5. Noise level and impact range of power gating induced P/G noise in 4*4-core MPSoC. agent in NoC

4. SENOC FOR MPSOC P/G NOISE ALLEVIATION

In this section, we will describe the mechanisms used by SENoC to alleviate the impacts of P/G noise during power gating. We formalize our approach by optimizing MPSoC performance under the task constraints and PU operation constraints for an MPSoC with *N* PUs and a queue of real-time tasks *Task* under P/G noise attacks.

4.1 SENoC Components

TM is in charge of the entire MPSoC, making system-wide decisions of task distribution and PU coordination. The states transfer of TM is shown in Fig. 6.

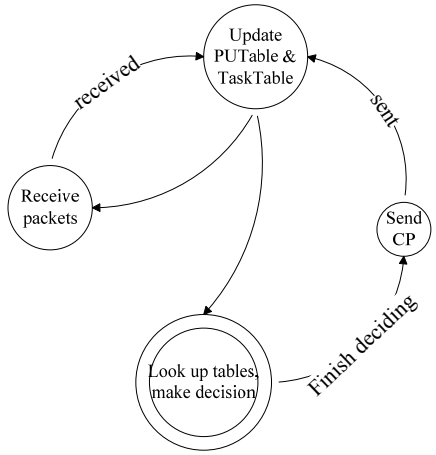

Figure 6. State transfer of TM

TM multicasts command packets to inform PUs of new tasks, and receives report packets and post-alert packets. Upon receiving a report packet or a post-alert packet, it updates Task Table and PU Table shown in Table 2 and Table 3. It should be noted that the Transient state of PU in PU Table generalizes all transient states, including *Idle* (*clock gated*), *ToOn*, *ToOff*, *ClkToOn*, and *ClkToOff* states in Table 1. TM is also in charge of the task queues. If Task queue is not empty, it selects one and a useable PU to carry on the task. All the tasks should fulfill the timing constraints. Then this information is packetized in a command packet which will then be multicasted through NoC.

Table 2. Task Table in Task Manager

Attributes	Illustrations
IntriRunTime	The intrisic run time for the task
RunTime	The actual run time for the task
MinStartTime	The minimum time for the task to start
StartTime	The actual start time for task
MinEndTime	The minimum end time for the task
EndTime	The actual end time for the task
PUID	Which PU is processing the task
State	Processing state of the task. 0: have not started; 1: started, including halting and finished
IsCritical	Whether the task is critical
RequireTime	Request time for the task
Index	The index in the task queue regarding topological structure of tasks

Table 3. PU Table in Task Manager

Attributes	Illustrations
Task	Which task the PU is processing
State	PU states, including Off, On, Active, Free, Transient
Useable	If a PU is Off or Free, it is useable
Impact range	A matrix indicating whether a PU will impact another PU
T_{on}, T_{off}	The time of power on, power off for a PU
Safe Voltage	The current safe voltage level suitable for a PU
Performance	Quality of the PU

Packets are generated by TM or node agents based on the run-time information provided by SENoC, transferred through NoC and analyzed by node agents. The detailed information of the four types of packets is listed in Table 4. In the table, valid receiver refers to those PUs or TM to which the packet information will affect their action.

For all the packets, a tag including information of packets' type, sender and receiver is attached. For command packet, a list of victims that TM calculated should also be attached. Therefore, a node agent who receives a packet will be capable to decide its next action together with the sensor information of the corresponding PU.

A node agent is capable of interpreting information in the packets, and extracting run-time information from different sensors according to some build-in lookup tables which are shown in Table 5. Additionally, node agents will generate packet with unified forms to ensure a reliable and secure packet transfer, and maintain the efficiency of MPSoC.

Table 4. Four Packets in SENoC

Packet type	Illuminations				
	Possible sender	Valid receiver	Transfer fashion	Generation condition	Changes in PU state
Command packet (CP)	TM	attacker, victims	multicast	task queue is not empty, useable PU exists	victims change to Transient
Alert packet (AP)	attacker	victims	multicast	the attacker powers off or on	attacker changes to Transient
Post-alert packet (P-AP)	attacker	TM,victims	multicast	the attacker is stable within safe voltage level	attacker changes to Active or Off
Report packet (RP)	attacker, victims	TM, attacker	unicast	when a PU transfers to a stable state	according to contents of the packet

Table 5. Table In Node Agent

Items	Illustrations	
	Information included	Function
Packetization synthesizer	the templates for synthesizing information to four types of packets	normalize the format of packets
Packet analyzer	different sorts of information in the packet and their connotations	decide the next action for local PU
Sensor analyzer	signals from different sensors	correspond to different run-time information, on-line adjustment

PU *p* becomes an attacker when a task is assigned to it, which indicates powering on; or when it is going to power off. Upon receiving the command packet, and understanding its role of an attacker after analyzing the command packet, PU *p* should know its victims and wait for their report packets. After receiving

the report packets from all its victims, it will multicast alert packet in the beginning of powering on, and multicast post-alert packet at the end of powering on, report its state of Free by unicasting report packet to TM when finishing a task. The state-transfer of an attacker is shown in Fig. 7.

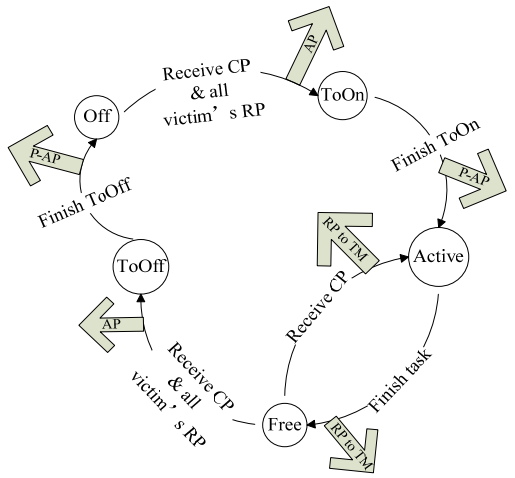

Figure 7. State tranfer of Attackers

PU *q* becomes a victim when it is operating and neighboring PU *p*'s powering on or off will affect it. In such case, for the sake of protecting its own process, *q* has to pause and go through *Idle* state until *p*'s powering action exerts no impact on *q*. Similar to *p*, *q* also experiences transient state *ClkToOff* when being protected, and *ClkToOn* when resurging. Report packets are sent to TM and attacker, reporting the state and action. The detailed information of state transfer and packets sending is shown in Fig. 8.

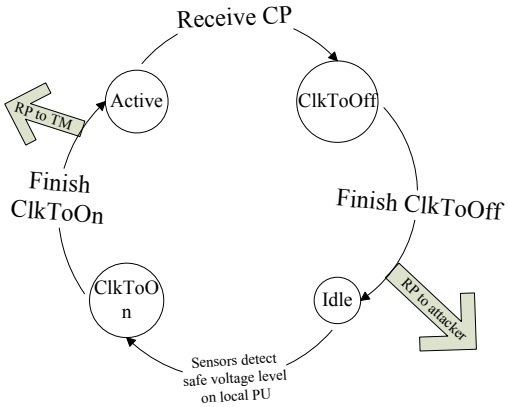

Figure 8. State tranfer of Victims

4.2 Overall Operation in SENoC

The operations of SENoC can be classified into the powering-on operation, and powering-off operation which are almost the same. We use the powering on operation as an example as follows. When there are tasks ready to process, TM selects Task *i* to operate:

1.1) If there is some PUs in Transient state, do nothing.

1.2) If there is no PU in Transient state, TM selects PU *p* to operate Task *i,* and calculate the theoretical operating time for later reference. This operation is as following:

1.2.1) If there is no *Free* or *Off* PU, do nothing.

1.2.2) If there is a *Free* PU, then this PU will be selected. TM packetizes a command packet to multicast, and updates PU Table. PU *p* carries out Task *i* after receiving and analyzing the command packet.

1.2.3) If there is an *Off* PU *p* with the least number of victims and relatively good *Performance*, PU *p* will be selected and becomes an attacker. A new *Safe Voltage* will be calculated by TM according to its *Performance* and previous *Safe Voltage*. TM multicasts command packet. After a node agent receives the command packet, it will know the role of itself, an attacker or a victim. Assume PU *q* is one of PU *p*'s victims, and it transfers to *ClkToOff* state after receiving command packet. During *ClkToOff*, it makes preparation for clock gating, including sending data to external memory (off-chip memory). Victim *q* unicasts report packet to attacker *p* after finishing such operations, transfers to *Idle* state during the same period.

After receiving report packets from all its victims, attacker *p* multicasts alert-packet, transfers to *ToOn* state during when it switches on sleep transistors and powers on. When its sensors report safe voltage level, a post-alert packet is generated and multicasted. A report packet will be sent to TM, then PU Table and Task Table are updated. Meanwhile, attacker *p* transfers to *Active* state and executes Task *i*. During the *Idle* state of victim *q*, it transfers to *ClkToOn* state after its own sensors detect *Safe Voltage* of attacker *p*. During the *ClkToOn* state, PU *q* prepares for resurging from clock gating, including loading relevant data from off-chip memory, etc. Victim *q* unicasts report packet to TM after finishing these actions, and at the meantime transfers to *Active* state, continuing the previous task. TM will update PU *q*'s information after receiving this report packet. After receiving all the report packets, TM will update items in PU Table and Task Table, including PUs' state and *Performance*, Task *i*'s *EndTime*, etc. And then TM will move on to the next action: assign a task or turn off a PU.

4.3 Task Scheduling Algorithms Used by TM

We adopt a Greedy Heuristic algorithm to perform dynamic task scheduling in MPSoC. In comparison to the on-line adjustment method, the corresponding stop-go algorithm is also implemented to simulate working process of an MPSoC without SENoC. The total execution time of a task T_{end}, and the times that the PU has been clock-gated (*CP*) and powered on/off (*PT*) during the execution of a task are all examined as parameters to measure the performance of MPSoC.

The on-line task scheduling algorithm (GH) is shown from line 4 to 24 in Fig. 9. This program simulates the process discussed in Section 4.2. The algorithm also includes the strategy for on-line task assignment and PU distribution. GH algorithm always runs the task with the earliest release time. If there are several tasks with the earliest *treq(i)*, GH chooses to always run the task with the longest IntriRunTime. When we choose a PU to execute the new task (D2 in line 7 of Fig. 10), GH gives first

priority to Free PUs. If there is no Free PU, consider Off PUs. If there are several Free PUs, GH chooses the PU with minimal $N_{PV}(p)$. If there are several Off PUs as candidates, GH always powers on the PU with the minimal $P_{on}(p,t)$ (total performance penalty to power on and off p). When it entails powering off, GH chooses to always power off the PU with the minimal $N_{PV}(p)$. If there are several PUs waiting for powering off (D3 in line 18 of Fig. 10), the selection rule of GH is similar to the rule of powering on a PU.

Greedy Heuristic (GH) algorithm for power gating induced P/G noise-aware on-line task assignment and scheduling

Input: the task set *Task* (including DAG *Link*, *treq (i)*, *length_p(i)*); the PU set *PU* (including the impact relation between PUs); $T_{onsettle}$, $T_{offsettle}$, $T_{ClkToOff}$, $T_{ClkToOn}$.

Output: T_{end}, CP, PT and the task assignment.

```
1  Initialize variables;
2  Time node t=0;
3  do
4    if there are requested tasks and their inputs are ready
5      D1: choose a task i to assign;
6      if timing constrains are satisfied
7        D2: choose a PU p to execute task i;
8        if available PU p exists
9          if chosen PU p is Off
10           if N_victim(p,t)>0 (PU p has some victims)
11             protect victims;
12             power on PU p to execute task i;
13           else
14             power on PU p to execute task i;
15         else
16           assign task i to PU p;
17   else if there are PUs waiting to power off
18     D3: choose a PU p to power off;
19     if timing constrains are satisfied
20       if N_victim(p,t)>0
21         protect victims;
22         power off PU p;
23       else
24         power off PU p;
25   t++;
26 while there is an un-finished task or an on PU
```

Figure 9. On-line gready heuristic algorithm for SENoC

GH's corresponding stop-go algorithm protects all the active PUs during powering on or off a PU, without concern of real voltage impact as GH does. Therefore, a system without SENoC can adopt stop-go algorithm, which is simpler and safer, but conservative according to our P/G noise model.

5. IMPLEMENTATION AND SIMULATION RESULTS

We first show a detailed time analysis for SENoC. Based on the analysis, simulations are performed to measure the performance of MPSoC with and without SENoC.

5.1 Time Consumption Analysis for SENoC

SENoC and PUs consume time in several ways, such as the signal transfer time and processing time (Table 6). Take the case shown in Fig. 5 as an example. Assume attacker p is the northwest PU in MPSoC, and victim q is on the right of it. Since there are 6 routers in the path from attacker p's node agent to TM , the time consumed for packets to transfer is $T_{trans}(pm) = T_R + T_{RR} + T_R + T_{RR} + T_R + T_{RR} + T_R + T_{RR} + T_R + T_{RR} + T_R = 6T_R + 5T_{RR}$. Upon receiving a packet, the node agent should analyze the contents and then send it to its PU through NI, and from NI to PU. This process consumes time $T_{receive} = T_{ana} + T_{RN} + T_{NP}$. For TM, it should also receive data through its interface with router, and analyze the packet, then change relevant information in PU Table and Task Table. This period is $T_{TMreceive} = T_{RT} + T_{ana} + T_{update}$. When a PU needs to send a packet, first it should send necessary information to node agent through NI, then node agent will packetize the information. This process consumes time $T_{send} = T_{NP} + T_{NR} + T_{pac}$. When TM needs to send a packet, it should packetize and send the package to the router beside it through its interface with the router. This period is $T_{TMsend} = T_{pac} + T_{RT}$. The ClkToOn period contains the time that node agent send relevant data to off-chip memory to save. This period is $T_{send} + T_{transfer} + T_{Rm}$, in which $T_{transfer}$ is the period of data transfer from PU to off-chip memory through routers on NoC and can be calculated the same way as $T_{trans}(pm)$ mentioned above. The period of loading data during ClkToOn can be derived in similar method.

Considering a series of operations and their time consumption regarding the powering-on operation, PU p is to be powered on and carry out a task. From that TM has selected task i for PU p and calculated p's victims, to that p is executing task i with all its victims resurged and TM has already known, this period of time can be calculated as follows. Consider the circumstance that there is no Free PU. TM selects task i and PU p to undertake i. This decision period is T_{dec}. Assume that $state(p)$ = Off. PU p becomes attacker and PU q is one of p's victims.

Fig. 10 shows the actions of TM, attacker p and victim q along with time. TM generates a command packet, which consumes T_{TMsend}, and multicasts it. Victim q transfers to ClkToOff state after receiving it, making preparation for clock gating, including sending data to off-chip memory to save, all of which consumes time $T_{trans}(qm) + T_{receive} + T_{ClkToOff}(q)$. Victim q unicasts report packet after finishing such operations, and transfers to Idle state during the same period, which consumes time $T_{trans}(pq) = TR + TRR + TR$ in this case. TM waits until report packet from all the victims have reached. Up to now, the total time is $\max\{ T_{trans}(qm) + T_{receive} + T_{ClkToOff}(q) + T_{send} + t_{trans}(pq) + T_{receive} \mid q \text{ is } p\text{'s victim} \}$.Then attacker p multicasts alert packet, which costs T_{send}. After that, attacker p transfers to ToOn state during which it powers on, including switching on sleep transistors within safe voltage level, transferring and loading

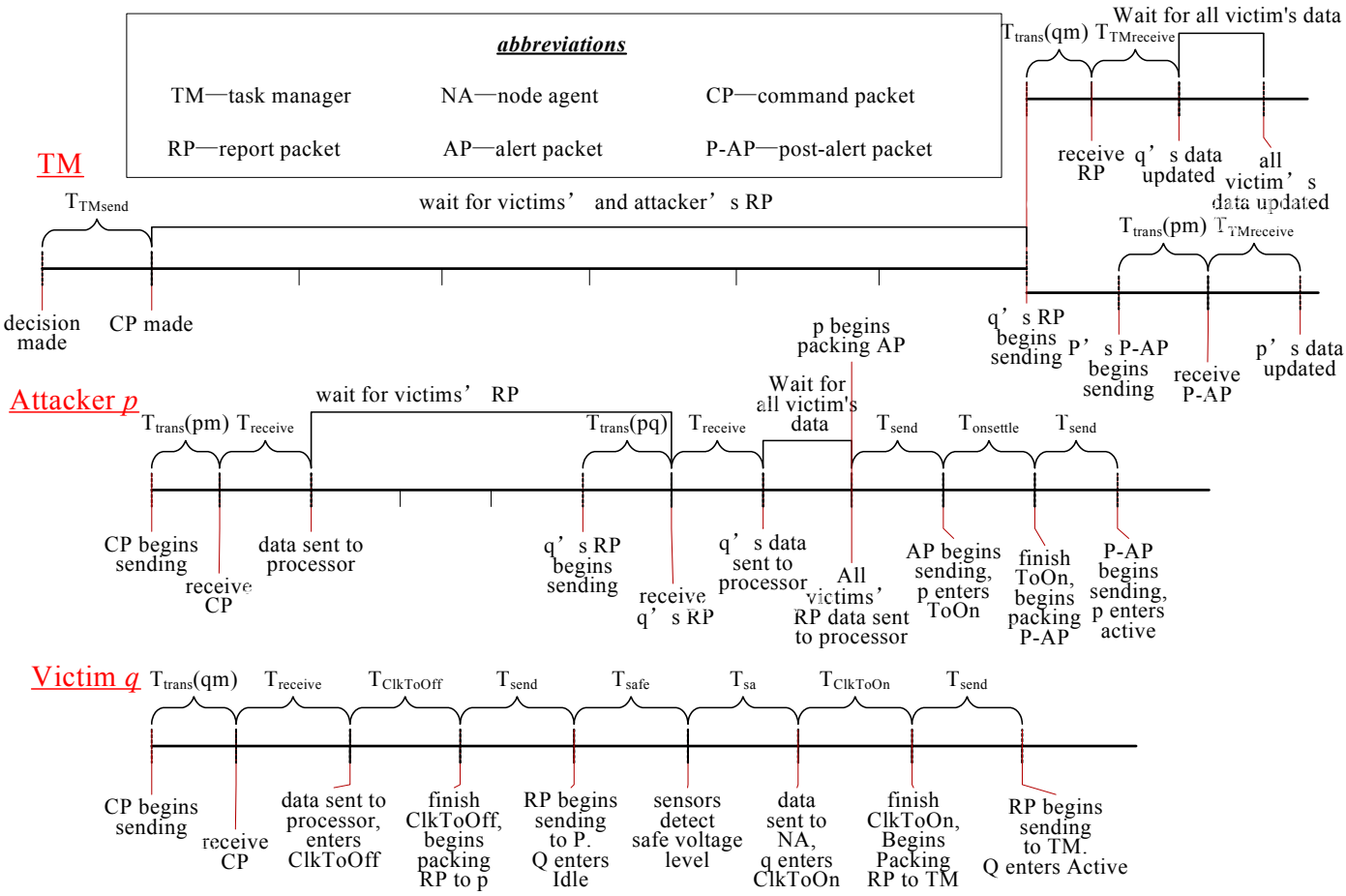

Figure 10. Actions of TM, Attacker *p* and Victim *q* in time

relevant data, etc. This process consumes time $T_{onsettle}(p)$. Then p multicasts post-alert packet, consuming it T_{send} and then enters Active state, beginning to execute task i. Post-alert transfers to TM, making it update lookup tables. This process consumes $T_{trans}(pm) + T_{TMreceive}$. The last part of TM's timeline suggests that there are two parallel operations: one dealing with victims, the other with attacker.

During the Idle state of victim q, its own sensors detect safe voltage level at local PU, and node agent is aware of that, consuming time $T_{safe}(q) + T_{sa}(q)$. It transfers to ClkToOn state. The ClkToOn state consumes $T_{ClkToOn}(q)$. Victim q unicasts report packet after finishing actions in ClkToOn. After packetization, it transfers to Active state, continuing the previous task. This report packet transfers to TM which will then update PU Table. This process consumes $T_{send} + T_{trans}(qm) + T_{TMreceive}$. Finally, from all victims begin clock gating, to all of them resurged and TM knows that, victims do not perturb each other, so this period should be max{ $T_{trans}(qm) + T_{receive} + T_{ClkToOff}(q) + T_{send} + T_{safe}(q) + T_{sa}(q) + T_{ClkToOn}(q) + T_{send} + T_{trans}(qm) + T_{TMreceive}$ | q is p's victim }.

Since the actions of attacker and victims do not interrupt each other, these two actions parallel that attacker enters ToOn state and that victims resurge to Active state after entering ClkToOff state. Therefore the total time for a whole process of powering-on is expressed as max { max { $T_{trans}(qm) + T_{receive} + T_{ClkToOff}(q) + T_{send} + T_{trans}(pq) + T_{receive}$ | q is p's victim } + $T_{send} + T_{onsettle}(p) + T_{send} + T_{trans}(pm) + T_{TMreceive}$, max { $T_{trans}(qm) + T_{receive} + T_{ClkOff}(q) + T_{send} + T_{safe}(q) + T_{sa}(q) + T_{ClkToOn}(q) + T_{send} + T_{trans}(qm) + T_{TMreceive}$ | q is p's victim }}. The period that TM updates the tables should be max{max{$T_{trans}(qm)+T_{TMreceive}$| q is p's victim}, $T_{trans}(pm)+T_{TMreceive}+\Delta t$ (p's P-AP begins sending, q's RP begins sending)}, in which Δt (p's P-AP begins sending, q's RP begins sending) represents the duration after q's RP begins sending and before p's P-AP begins sending.

5.2 Implementation and Simulation Setup

The simulations are based on the MPSoC P/G noise simulation and analysis platform. The simulations assume that the average power consumption of a single PU is 30mW, and the area of a single PU is 660μm×660μm. Based on SPICE simulation results using 45nm standard logic cells, the noise toleration of V_{dd}-V_{ss} is set to be 100mV, and hence V_{safe} is set to be 700mV. The corresponding R_{impact} of each attacker p is derived for 4*4-core to 8*8-core MPSoCs. The P/G network RLC parameters are obtained from PTM interconnect model [14].

We adopt four basic topological structures of tasks to make comparison: (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). What is more, for task number of 60 and 80, we adopt two sub-structures for each of SP,

TT, and *FC*: structure expanded by depth (designated as *a*) and by width (designated as *b*), from 40-task structures.

5.3 Simulation Results

In our simulation, we test MPSoC using SENoC and stop-go method for 6 to 80 tasks with different task structures. The actual execution time is longer than ideal execution time for all the cases in our study. Difference in task structures, MPSoC scales, task numbers and algorithms affects execution time differently. We define r_{ai} (SENoC)= (T_{end} (SENoC)- T_{end_i}) / T_{end_i} to measure the efficiency of the method with SENoC, in which T_{end_i} denotes the ideal finih time for all the tasks assuming that power gating is not adopted. To compare different methods directly, we define r_{Gs} = (T_{end}*(stop-go)* - T_{end}*(SENoC)*) / T_{end}*(stop-go)* to compare SENoC method and stop-go method. r_{ai}, r_{Gs}, *CP* and *PT* for different task structures in 8*8 MPSoC are listed in Table 7.

Table 6. Types of Time Consumption

Time	Illustrations	
	Start point	**Terminal**
T_{RR}	router A	A's neighboring router B
T_{NR}	PU's NI	nearest router
T_{RN}	router	nearest PU's NI
T_{sa}	sensor	local node agent
T_{Rm}	router's interface with memory	memory
	memory	router's interface with memory
T_{RT}	router beside TM	TM
	TM	router beside TM
T_{NP}	NI	PU
	PU	NI
T_{pac}	packetization time	
T_{ana}	packet analyzing time	
T_R	the period for a router to make decision	
T_{safe}	the period from a victim enters Idles state to its sensors detect safe voltage level	
T_{safe}^{p}	the period from attacker *p* powers on to its voltage level is stable within safe range	
$T_{ClkToOff}(q)$	the time *q* costs during ClkToOff state	
$t_{ClkToOn}(q)$	the time *q* costs during ClkToOn state	
$T_{onsettle}(p)$	the time *p* costs during ToOn state	
T_{update}	the time that TM updates information in PU Table and Task Table	

Table 7 shows that MPSoC with SENoC achieves impressive T_{end} improvement compared with MPSoC without SENoC. Especially, the improvement is obvious for task structures that many tasks run in parallel like $TASK_{NC}$. For 8*8-core MPSoC, on average, we have r_{Gs} = 26.20%, which shows that SENoC helps to achieve an average performance improvement of 26.20%. We estimate the area overhead based on the 45nm design and the NoC simulations using NS2 [16]. The area overhead of the SENoC for 8*8-core MPSoC are 1.4%.

Table 7. Performance of 8*8-core MPSoC (CP: Clock Gated Times; PT: Power On/Off Times)

#Task	Task structure	SENoC			Stop-go		r_{Gs1}(%)
		r_{ai} (%)	CP	PT	CP	PT	
6	NC	2.758	10	12	30	12	17.150
8	NC	3.1143	20	16	56	16	29.845
10	NC	1.6253	32	20	90	20	30.522
12	NC	1.4768	39	24	132	24	32.319
20	NC	2.4386	90	40	380	40	46.151
40	NC	3.4001	196	80	1560	80	59.613
40	SP	0.8987	22	18	56	16	7.434
40	TT	1.0313	146	54	450	54	26.371
40	FC	1.0991	34	28	114	28	11.999
60	NC	1.2925	285	120	3540	120	68.254
60	SP_a	0.8803	28	20	56	16	5.075
60	SP_b	2.37	46	28	132	24	9.173
60	TT_a	1.3756	233	84	958	179	29.998
60	TT_b	7.6552	276	90	1312	189	31.912
60	FC_a	0.8923	56	38	156	36	10.206
60	FC_b	2.4025	95	44	262	38	14.863
80	NC	95.236	320	128	4032	128	43.767
80	SP_a	0.7593	30	20	56	16	3.771
80	SP_b	0.7966	91	38	30	12	14.296
80	TT_a	1.453	385	126	56	16	35.332
80	TT_b	0.7782	382	128	90	20	40.239
80	FC_a	0.934	90	56	132	24	12.719
80	FC_b	1.0342	169	62	380	40	21.552

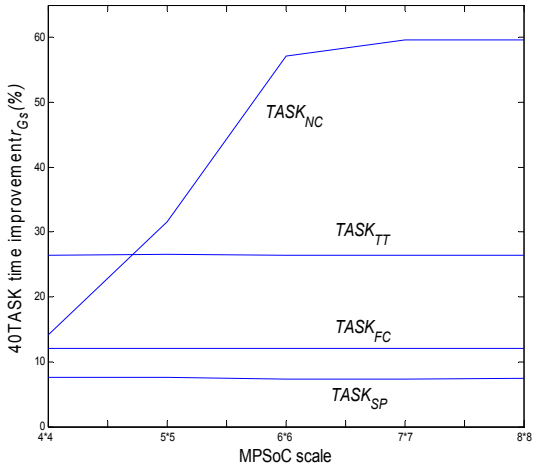

Figure 11. Performance improvement of MPSoC using SENoC with 40 tasks under different task structures and

Fig. 11 shows the performance improvement of MPSoC using SENoC with 40 tasks under different task structures and MPSoC scales. The performance improvement increases dramatically as the MPSoC scale increases in $TASK_{NC}$. However, for task structures NC, FC and TT, the increase in performance improvement is not so obvious. This is because for $TASK_{NC}$, tasks

can run in parallel. Generally, the trend of increase in performance improvement slows down when MPSoC scale increases. This is due to the fact that a relatively small number of PUs can handle the given set of tasks.

Besides, CP and PT while using SENoC decreases drastically in most cases. This indicates SENoC helps to decrease the time of power off and on, thus effectively avoid powering on/off induced P-G noise. Therefore, MPSoC will work more safely with higher reliability when using SENoC.

6. CONCLUSIONS

This paper proposes a systematic approach, on-chip sensor network (SENoC), which not only detects reliability threats and shares related information among PUs, but also plans and coordinates the reactions of related PUs in MPSoC. SENoC is integrated with NoC to ensure that critical information and decision is delivered in a timely fashion. SENoC is used in our case study to alleviate the impacts of simultaneous switching noise in MPSoC's P/G network during power gating. Based on the detailed noise behaviors under different scenarios derived by our circuit-level MPSoC P/G noise simulation and analysis platform, the case study shows that SENoC helps to achieve on average 22.2% performance improvement compared with the traditional stop-go method with 1.4% area overhead in an 8*8-core MPSoC in 45nm.

7. ACKNOWLEDGMENTS

The authors are grateful to the reviewers, who offer us helpful suggestions to improve this paper. This work is partially supported by National Natural Science Foundation of China (No.60870001), 863 project (No. 2009AA01Z130), and partially by Hong Kong SAR RGC Earmarked Research Grant 621108 and Hong Kong University of Science and Technology PDF.

8. REFERENCES

[1] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. IEEE Journal of Solid-State Circuits, 43(1):29–41, 2008.

[2] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, Matthew Mattina, Chyi-Chang Miao, Carl Ramey, David Wentzlaff, Walker Anderson, Ethan Berger, Nat Fairbanks, Durlov Khan, Froilan Montenegro, Jay Stickney, and John Zook. Tile64tm processor: A 64-core soc with mesh interconnect. In Proc. Digest of Technical Papers. IEEE International Solid-State Circuits Conference ISSCC 2008, pages 88–598, 2008.

[3] Shekhar Borkar. Thousand core chips: a technology perspective. In DAC '07: Proceedings of the 44th annual conference on Design automation, pages 746–749, New York, NY, USA, 2007. ACM.

[4] V. Petrescu, M. Pelgrom, H. Veendrick, P. Pavithran, and J. Wieling, "Monitors for a signal integrity measurement system," Solid-State Circuits Conference, 2006. ESSCIRC 2006. Proceedings of the 32nd European, pp. 122–125, Sept. 2006.

[5] C. Poirier, R. McGowen, C. Bostak, and S. Naffziger, "Power and temperature control on a 90nm itanium-family processor," Solid-State Circuits Conference, 2005. Digest of Technical Papers. ISSCC. 2005 IEEE International, pp. 304–305 Vol. 1, Feb. 2005.

[6] R. McGowen, C. Poirier, C. Bostak, J. Ignowski, M. Millican, W. Parks, and S. Naffziger, "Power and temperature control on a 90-nm itanium family processor," Solid-State Circuits, IEEE Journal of, vol. 41, no. 1, pp. 229–237, Jan. 2006.

[7] K. Sohn, N. Cho, H. Kim, K. Kim, H.-S. Mo, Y.-H. Suh, H.-G. Byun, and H.-J. Yoo, "An autonomous sram with on-chip sensors in an 80nm double stacked cell technology," VLSI Circuits, 2005. Digest of Technical Papers. 2005 Symposium on, pp. 232–235, June 2005.

[8] K. Sohn, H.-S. Mo, Y.-H. Suh, H.-G. Byun, and H.-J. Yoo, "An autonomous sram with on-chip sensors in an 80-nm double stacked cell technology," Solid-State Circuits, IEEE Journal of, vol. 41, no. 4, pp. 823–830, April 2006.

[9] C. Chan, Y. Chang, H. Ho, and H. Chiueh, "A thermal-aware power management soft-ip for platform-based soc designs," System-on-Chip, 2004. Proceedings. 2004 International Symposium on, pp. 181–184, Nov. 2004.

[10] Alexander Wei Yin, Liang Guang, Pasi Liljeberg, Pekka Rantala, Ethiopia Nigussie, Jouni Isoaho, Hannu Tenhunen. Hierarchical Agent Architecture for Scalable NoC Design with Online Monitoring Services Proceedings of MICRO 41

[11] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen. An event-based network-on-chip monitoring service. In Proc. of the 9th IEEE International High-Level Design Validation and Test Workshop, pages 149–154, 2004.

[12] C. Ciordas, K. Goossens, A. Radulescu, and T. Basten. Noc monitoring: impact on the design flow. In Proc. IEEE International Symposium on Circuits and Systems ISCAS 2006, pages 1981–1984, 2006.

[13] Yan Xu, Weichen Liu, Yu Wang, Jiang Xu, Xiaoming Chen, Huazhong Yang, "On-line MPSoC Scheduling Considering Power Gating Induced Power/Ground Noise", in ISVLSI 2009, Tampa, USA, pp. 109-114.

[14] Nanoscale Integration and Modeling (NIMO) Group, ASU. Predictive Technology Model (PTM). [Online]. Available: http://www.eas.asu.edu/˜ptm/

[15] Nangate Open Cell Library. [Online]. Available: http: //www. opencelllibrary.org

[16] NS2, http://nsnam.isi.edu/nsnam