# PowerV*i*P: SoC Power Estimation Framework at Transaction Level

Ikhwan Lee[1], Hyunsuk Kim[1], Peng Yang[1], Sungjoo Yoo[1], Eui-Young Chung[2], Kyu-Myung Choi[1], Jeong-Taek Kong[1], and Soo-Kwan Eo[1]

[1]CAE center, System LSI division, Semiconductor Business, Samsung Electronics, Co. Ltd.
San 24 Nongseo-Dong, Giheung-Gu, Youngin, Gyeonggo-Do, 449-711, Korea
e-mail: {ikhwan.lee, hyunsuk71.kim, peng.yang, sungjoo.yoo, kmchoi, jkong, sookwan.eo}@samsung.com

[2]School of Electrical and Electronic Engineering, Yonsei University
134 Sinchon-Dong, Seodaemun-Gu, Seoul, 120-749, Korea
e-mail: eychung@yonsei.ac.kr

**Abstract - In this work, we propose a SoC power estimation framework built on our system-level[1] simulation environment. Our framework provides designers with the system-level power profile in a cycle-accurate manner. We target the framework to run fast and accurately, which is enabled by adopting different modeling techniques depending on the power characteristics of various IP blocks. The framework can be applied to any target SoC design.**

## I. Introduction

System-level design paradigm has been widely adopted to cope with the ever-increasing complexity of System-on-Chip (SoC) design. The high simulation speed at the system level allows designers to explore the huge design space of modern SoC designs.

Design space exploration of modern SoC devices usually deals with three design constraints: performance, area and power consumption. The former two constraints have been relatively well understood in the traditional design flow. However, the market needs for low power devices have introduced the third constraint, power consumption. Since typical SoC devices have many components heavily interacting with each other, it is essential to examine the power consumption of each component in the system context [1]. The power profile generated by independent simulation of each component may mislead designers to a local power optimal design. This means that power estimation should also be performed at the system level.

In order to perform system-level power estimation, we need to build power models of all components. However, a salient feature of SoC is that it has many heterogeneous components with varying power characteristics, ranging from very regular structures such as on-chip SRAM to irregular custom IP blocks such as video codec. This makes it extremely difficult to derive a single modeling methodology that can cover every component constituting a SoC device. Thus, different approaches are adopted for different components. In any cases, however, we need to consider the relationship among the following three factors: simulation speed, estimation accuracy and modeling effort.

Estimation accuracy is often compromised for simulation speed and modeling effort. By exploiting the heterogeneity of SoC, we can make a good trade-off among them. For example, components with little power variation can employ simple power models to reduce the modeling effort while boosting the simulation speed. For custom IP blocks, we also need to take into account the effort to build the system-level simulation models. Unlike processor cores and bus fabrics whose models are provided by the vendors, legacy custom IP blocks that exist in the form of RTL usually do not have system-level models.

In this paper, we present a system-level power estimation framework, PowerV*i*P, built on our system-level simulation framework, V*i*P [2]. In PowerV*i*P, different power modeling techniques are employed for each component: processor cores, bus fabrics, custom IP blocks and memories. Moreover, for custom IP blocks, an RTL to V*i*P model translation technique is adopted to reduce the modeling effort. PowerV*i*P provides designers with useful power information fast and accurately as well as easy modeling capability.

The paper is organized as follows. Section II describes related work and Section III presents our contributions. The details on power modeling of the processor cores, bus fabrics, and custom IP blocks are presented in Section IV, V, and VI, respectively. The memory power model used in this work is briefly described in Section VII. Section VIII presents the integrated framework based on the separately modeled and validated components. Section IX concludes the paper.

## II. Related Work

Extensive studies dealing with the problem of power estimation have been proposed, ranging from circuit-level modeling to behavioral modeling approaches [1, 3]. While highest accuracy is achieved at the lowest level, estimation speed degrades significantly as we move down to lower levels. Therefore, it is crucial to derive a method that performs the best trade-off between estimation accuracy and speed.

Co-simulation based approach is one way to achieve a

[1] In this paper, we will use the term transaction level and system level interchangeably to represent their union.

0-7803-9451-8/06/$20.00 

good trade-off between accuracy and speed. In [1], multiple power simulation engines work in a concurrent and synchronous manner. In an effort to minimize the speed degradation caused by co-simulation, they propose several speed-up techniques. In our work, we adopt a model translation technique to completely eliminate the overhead of co-simulation.

A dynamic power model selection scheme at the system level is proposed in [3], where computation effort among different SoC components is allocated at run-time for the best estimation time and accuracy trade-off.

Most of the work on power modeling has focused on power modeling of individual components such as processors, bus fabrics, memories, and custom IP blocks.

The first work on processor power modeling is reported in [4]. Their model quantifies instruction base energy and inter-instruction energy effects to enable fast software energy estimation. Wattch [5] and SimplePower [6] are two well-known power estimation tools in academia. A power model tailored for the Intel XScale processor is proposed in [7]. Their power model is based on module activities, where each module has its power equation embedded in Sim-XScale simulator. The power equations are constructed using transistor level schematics of functional units and a high-level view of transistor gate and drain capacitances. A software power estimation tool, JouleTrack, is presented in [8]. They propose a power characterization methodology that avoids explicit power characterization for each differentiated instruction class.

Bus system power modeling and estimation has been addressed in many different flavors, from the simple analytic model to the detailed gate-level switching activity based model [9 - 12]. Several papers are published to address the problem at a higher abstraction level, the system level [13 - 15]. In practice, most current commercial design flows utilize RTL and gate-level power estimation tools. However, due to their poor efficiency, it is impractical to apply them at the early stage of design, when many different architecture options have to be explored. Most of the work considers only the global wire, which is comparatively easy to model, but not the communication architecture components. This is incomplete because, as pointed out in [16], for complex communication network, the global wire only contributes a small potion of total power consumption.

Most of the existing work on IP power modeling takes RTL level approach. A few suggest behavioral-level methods; however, their accuracy is too low because of the mismatch between the behavioral description and the real implementation. The estimation accuracy becomes even lower when we employ an analytical method. Therefore, significant amount of work has been done on RTL macro modeling of IP power consumption [17]. Although the procedure to build an RTL power macro model is clear, it is still difficult to automate the process.

## III. Contributions

System-level simulation at the early design phase has become essential to search for optimal system architecture and also to enable early software development. We have developed a cycle-based simulation framework, called V*i*P, which can perform concurrent, cycle-accurate and synchronous system-level simulation [2].

On top of V*i*P, we build power models for each major SoC component to estimate the power consumption as well as the performance in a synchronous manner. Note that since the V*i*P framework provides synchronous activity information, the power models can provide more accurate power numbers in the system context [1].

Our goal in PowerV*i*P development is three-fold:

- maintain the target accuracy level
- keep the simulation overhead incurred by power estimation low
- make the power models easy to be customized to an arbitrary target design, since it is used at the system-level design phase where architecture exploration is performed

To achieve the conflicting goal of building a fast, accurate and easy-to-build power model, we take a component-based approach. As shown in Fig. 1, heterogeneous components of a SoC can be categorized into processors, bus fabrics, custom IP blocks, and memories. To achieve the best trade-off between simulation speed and accuracy, we apply different modeling techniques for each component depending on the power characteristics.

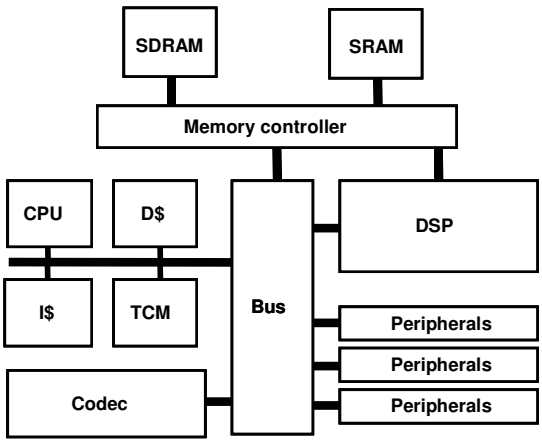


Fig. 1. A System-on-Chip design.

The procedure to build PowerV*i*P follows three steps in general. First, we set up a gate-level or RTL power analysis environment per IP component to extract (characterize) its power values. Next, we build a power model with the extracted power numbers. Finally, we annotate the power model into the system-level model of the IP component to generate power numbers during system-level simulation. For seamless adaptation of the power model to technology transitions, e.g., 130nm to 90nm or high speed to low power process, we automate this process by provisioning scripts that perform the power characterization and model parameter extraction steps.

In the following sections, we propose new approaches

applied to a SoC platform in detail in the following order: processor cores, bus fabrics, custom IP blocks, and memories. Power model development procedure and its validation result are separately presented in each section.

## IV. ARM926EJ-S Processor Power Model Development

The ARM926EJ-S processor is widely adopted in SoC as a controller as well as a small data processing engine; thus, we first embark on power modeling of the processor. Currently, power modeling of the ARM1176 processor is being conducted using the methodology described herein.

### *A. ARM926EJ-S architecture*

The ARM926EJ-S processor has a five stage pipelined data path and a Harvard cache architecture. The size of the caches can be from 4KB to 128KB. The ARM926EJ-S processor also has a fill buffer (FB) that keeps the most recently fetched cache line.

In the ARM926EJ-S processor, any instruction that modifies the program counter (such as a branch, or 'MOV pc, r0') causes a non-sequential instruction accesses on the next cycle. An instruction access by 'PC increment by 4' that crosses the cache line boundary also causes a non-sequential access. In Fig. 2 (a), a non-sequential (NS) access causes all four cache tag memories and data memories to be accessed along with the fill buffer. Whereas a sequential access (SEQ) causes only the data memory where the data is located is accessed as in Fig. 2 (b). In Fig. 2 (c), if the data is accessed from the fill buffer, there in no access to the cache.

For data caches, load multiple (LDM) and store multiple (STM) instructions support sequential accesses. LDR and STR instructions incur non-sequential accesses.

### *B. ARM926EJ-S power states*

We separate the processor power model into two parts: Processor core model and cache model. This separation comes from two observations. One is that caches can be configured differently (in terms of size, associativity, etc.) for various applications. Thus, one single model will not give an accurate estimation. The other observation is that the power consumption of caches gives a large variation. In the ARM926EJ-S processor, the cache power consumption ranges from 3% up to 60% of the total power. Therefore, we decide to model the core logic block and cache memory separately.

**Processor core: two simple power states**

We observe that the core logic can be in one of the two states: busy state and idle state (stalled by interlocks). There are numerous studies on processor power modeling, where more complex instruction level power states are identified [4, 7, 8]. However, in our work, we find that the two-state core power model gives more than 95% of the core power estimation accuracy for all of our benchmarks. On the other hand, one state model performs very poorly with its accuracy level of less than 70% for some benchmarks. Thus, we adopt the two-state power model for the processor core.

**Activity-based coarse-grain cache power model**

Most of the previous work on cache power modeling has exploited circuit-level information such as bit line and word line capacitive loads to generate flexible cache power models [5-7]. In industry, cache memories use memory compiler-generated SRAMs, where power values for each module are also provided for each type of read and write access. Thus, our cache power is modeled as a sum of power values for all accessed SRAM modules. For SRAM modules not accessed during the cycle, their static power values are added.

The ARM926EJ-S cache access behavior can be categorized into three different types as shown in Fig. 2. In power perspective, a non-sequential access consumes more than four times of power than a sequential access, since the cache power is the sum of dynamic power of all activated modules (tag memories and data memories) and static power of inactive modules. It dictates that in the ARM926EJ-S caches, non-sequential accesses and sequential accesses should be differentiated for accurate power estimation.

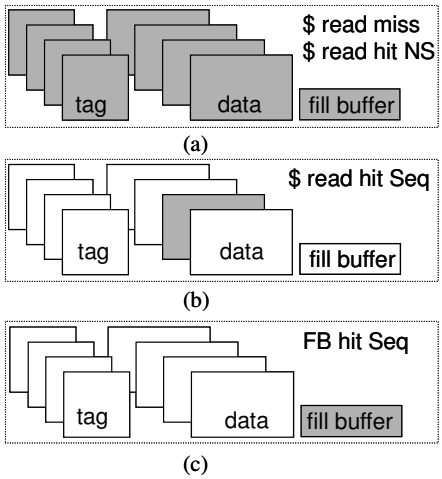


Fig. 2. ARM926EJ-S cache activity patterns.

Table I lists our identified data cache states and their corresponding module activities and power equations. In the table, *Tr* (*Tw*) and *Dr* (*Dw*) represent module power numbers for *Tag read* (*Tag write*) and *Data read* (*Data write*), respectively, obtained from our in-house memory compiler. The states are identical for the instruction cache except that there is no cache write hit or miss states. In this work, we ignore the power consumed by fill buffers.

Instructions and data are accessed from the fill buffer until it is evicted to the cache in two cycles (as shown 1st write-back and 2nd write-back in Table I) by the following cache line fetched in from the bus. Instruction fill buffer (I-FB) hit counts accounts for approximately 10% of the instruction cache hit counts in our *dhrystone* benchmark. If

an instruction fill buffer hit is encountered and the PC increments by 4, then it is I-FB sequential read, where negligible amount of power is consumed by the fill buffer. Therefore, it should be distinguished if the data is read from the cache or fill buffer to estimate power accurately.

TABLE I. Activity-based cache power model.

| Cache states | Module activity | Power Equation |
|---|---|---|
| sequential (cache) read | 1 data read | Dr |
| non-sequential (cache) read / read miss | 4 tag reads and 4 data reads | Tr*4+ Dr*4 |
| Data cache write hit | 4 tag reads, 1 tag write and 1 data write | Tr*4+Tw +Dw |
| Data cache write miss | 4 tag reads | Tr*4 |
| FB -> cache write (1st write-back) | 1 tag write and 4 data writes | Tw+Dw* 4 |
| FB -> cache write (2nd write-back) | 4 data writes | Dw*4 |
| sequential FB read | - | - |

### C. *Power (re-)characterization flow*

Power consumption is a complex function of many parameters. Depending on the quality of implementation, the same RTL can result in very different power values at the gate-level netlist. For example, two of our sample designs of the ARM926EJ-S show as much as twice power difference at the same frequency level, even though they are implemented with the same technology library. This implies that '*characterize once'* approach might not hold true in real applications.

In general, power characterization at the gate level proceeds as follows: (1) obtain the signal toggle information from gate-level simulation, (2) estimate the gate-level power from the toggle information using power libraries, and (3) calculate per-state power values using the estimated power information. If the power characterization is performed manually for each different gate-level netlist, it will be long, tedious, and error-prone task.

To reduce the characterization efforts, we set up an automated characterization flow as shown in Fig. 3, where designers can characterize power values repeatedly without investing much effort. The characterized power values are simply read by our simulator annotated with the power model to produce software power profiles. Note that the power model itself does not need any modification. We find that the power model itself is valid for different implementations of the same RTL.

Fig. 3 shows our power characterization flow. We first build a gate-level and RTL co-simulation template, where an RTL testbench with a simple bus and memory module drives the simulation with the ARM926EJ-S gate-level netlist of interest to generate the cycle-by-cycle signal toggle information as well as signal traces to infer the power states, using *dhrystone* benchmark. The toggle information is then fed into our in-house gate-level power estimation tool to generate cycle-by-cycle power values. The per-state power value is obtained by averaging the estimated cycle-by-cycle power values. All the aforementioned steps are performed automatically without any user intervention. The obtained per-state power value is finally annotated into our power simulator. We use the characterization flow to obtain the core power states in our power model. Note that the cache power model is activity-based and its SRAM module power value is provided by our memory compiler.

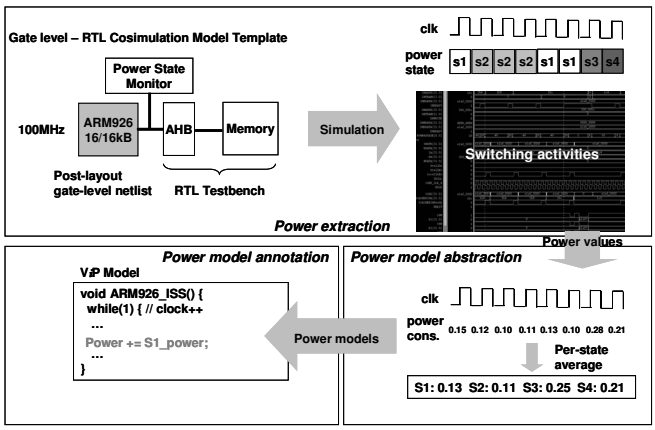


Fig. 3. Power characterization flow.

### D. *Power model validation*

Our simulation with five benchmarks shows 93%~98% of average power estimation accuracy. Fig. 4 shows cycle-by-cycle estimation result for a short code segment. It can be seen that the estimated power values closely track the power values measured at the gate-level. Regarding the power estimation speed of our simulator, it performs approximately 1600 times faster than gate-level estimation.

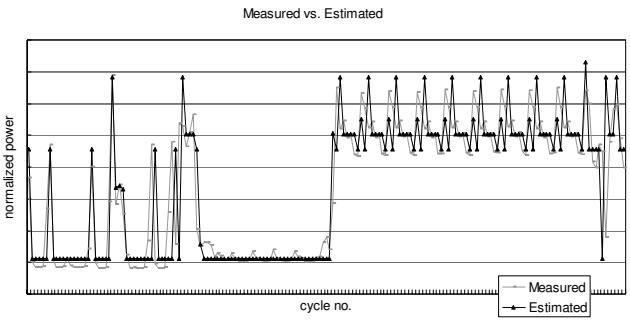


Fig. 4. Cycle-by-cycle estimation accuracy.

## Ⅴ. Component-based Transaction-level Power Modeling for ARM AXI Bus System

The development of today's semiconductor technology provides unprecedented computing speed that is shifting the IC design bottleneck from computation capacity to communication bandwidth and flexibility. The global communication becomes so difficult that more and more

designs turn to SoC architecture where a set of local blocks are connected with a communication network. Recent research [19] shows that the on-chip interconnect architecture not only has significant impact on the system performance and energy efficiency, it is also a significant source of power consumption, which is still increasing with the complexity of the system. Managing and optimizing power of this important SoC component require a detailed understanding of its characteristics.

### *A. AXI Bus architecture*

The AMBA AXI 3.0 protocol is targeted at high-performance, high-frequency system designs and includes a number of features that make it suitable for a high-speed sub-micron interconnects.

AXI 3.0 supports simultaneous read and write accesses. Both READ and WRITE transactions have their independent address and data channel. It also supports split transaction, which means the address/command phase and the data phase are separated. In addition, if configured to use ID, the AXI 3.0 compliant masters will assign an ID number to each outgoing transaction and check the ID of each in-coming response. This enables the implementation of multiple outstanding transactions and out-of-order transaction completion, which provide efficient communication for a wider variety of slaves.

The AXI 3.0 protocol only defines the interface between a master and a slave. To connect multiple masters and slaves, ARM suggests the new bus architecture, ARM Configurable Interconnect (ACI). Different from its predecessor AMBA AHB, ACI uses crossbar architecture to provide much higher communication bandwidth. Whenever there is no conflict, e.g., two masters sending commands to one slave at the same cycle, the transactions issued by one master will not be interfered or obstructed by other masters.

### *B. Power modeling*

What ACI differs from other bus system, e.g., AHB, is its crossbar architecture, which allows parallel bus accesses from multiple masters and introduces complex competition and coupling effects. For instance, the power consumption of one master at one specific cycle is not only decided by the communication between this master and the bus, but also depends on the communications conducted by other masters. To make it even worse, each master can do READ and WRITE in parallel. All these result in a huge number of states which are needed to represent the behavior of the bus.

To counter the problem, we have developed our transaction level bus power estimation method using a component based approach. The crossbar consists of components such as decoders, routers and arbitrators. The basic idea is to identify, at each basic state, which component is active and how much power it consumes. At run time, the bus state can be looked as a combination of these basic states, e.g., master 1 is decoding the address while master 2 is sending write data. Hence, to model all the bus states, we need only to characterize the limited number of basic states.

**Power model extraction**

The model extraction environment is shown in Fig. 5. We use the ARM ACI toolset to configure the bus interconnect, generate the RTL code and synthesize the circuit. Then the gate-level RTL code is simulated and an in-house tool is used to collect the gate-level switching activity information. Based on this information, the gate-level power number could be collected. The Cadence eVC environment is used to feed in random test sequences into the calibration flow.

Extraction of the power model of each component:

Given an ACI bus configuration, i.e., the number of masters, the number of slaves and other parameters such as data width, we synthesize the gate-level RTL code for our flow. During this step, we consider only the case when one master is communicating with one slave and configure the eVC to generate 8 random test sequences:

- READ with burst length of 1, 4, 8 and 16 words, respectively
- WRITE with burst length of 1, 4, 8 and 16 words, respectively

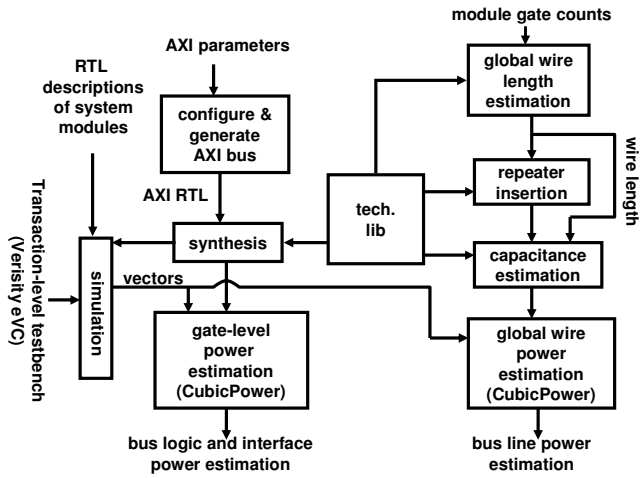


Fig. 5. The power model extraction environment.

This will rule out the coupling between multiple parallel transactions and allow us to focus on the behavior of each ACI component under all the basic transaction states.

Coupling effect:

To account for the coupling effect caused by multiple masters and slaves, another set of coefficients are introduced. We separate the coupling contribution into two items, one is related to the number of bursts the bus transferred during a period, and the other is related to the number of active cycles in that period.

With our eVC automatic test sequence generation environment, we systematically vary the number of active masters and slaves, and calibrate the gate-level bus power consumption under each situation. From these results, linear regression is applied to extract the coupling coefficients.

**Power estimation**

After extracting the transaction level bus power model, we apply the model to a sequence of bus transactions to estimate the bus power consumption of the bus during that period. The input can be at both the transaction level and the signal level.

For each bus cycle, we count the number of READ related transactions and the number of WRITE related transactions being performed by the bus. From them, we could know the specific state of each component and their basic power number accordingly. Thus we obtain the basic power estimation, without considering the coupling effect. To compensate the coupling effect when there are more than one active master and slave, a simple linear equation is applied. By accumulating the power estimation of each bus cycle for a period of time that we are interested in, we could finally obtain the total power consumption during that period.

### C. *Model validation*

In this section, we validate our model by presenting the estimation result of our transaction level power model with various bus configurations. Our experiment set-up is as follows. Given a configuration, such as frequency, data bandwidth, the number of masters, and the number of slaves, we use Synopsys Design Compiler to synthesize the bus architecture. Then our model extraction flow is applied to extract all the coefficients, based on the gate-level simulation result.

Fig. 6 summarizes our result for a 4x4 ACI bus, i.e., a configuration with four masters and four slaves. Due to the large amount of data, we present only the result when the burst length is 8 words. The other results with different burst length show similar characteristics.

In Fig. 6, the horizontal axis shows the number of active masters and slaves, which represents how many masters are really communicating with how many slaves. The vertical axis gives the estimation error of our transaction-level power model from the gate-level simulation. The results of the estimation with coupling effect and without coupling effect are separately plotted as well as the type (read/write) of the transactions. We can see clearly that the estimation error is significantly reduced if the coupling effect is taken into account. The maximum error drops from 16.24% to 5.57%, and the average error drops from 6.23% to 1.87%.

We have applied our power model to four different bus configurations. For all our experiments, the maximum error against the gate-level estimation is less than 10%, and on the average, it is below 5%.

The power model integrated into the ViP runs more than 100 times faster, compared with gate-level power estimation.

## VI. Automatic Generation of Power-Annotated ViP Models for Custom IP Blocks

Power modeling and estimation of custom IP blocks encounter two major challenges. First, a large amount of

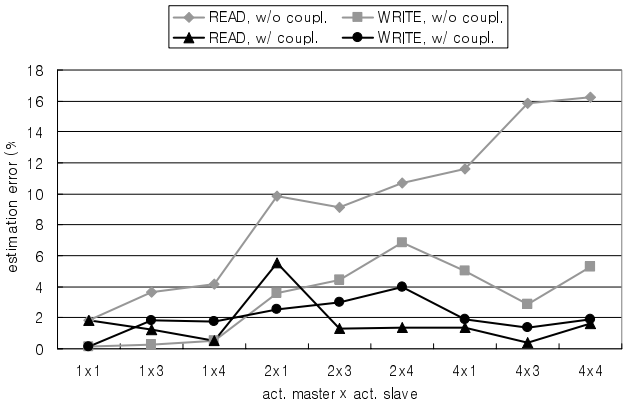


Fig. 6. Power estimation result, burst length 8.

work is required to build the high-level simulation model. This becomes critical when we consider frequently updated IP blocks with low reusability. Second, IP blocks with irregular power characteristics make it difficult to characterize their power consumption.

We propose a practical technique to meet the challenges for custom IP blocks. Our target is to perform a good trade-off between estimation accuracy and speed while minimizing the effort spent on power and ViP modeling. High accuracy is achieved by employing RTL power macro modeling method, and the speed overhead of co-simulation is completely eliminated by using automatic RTL to ViP model translation technique. The automatic RTL to ViP model translation technique additionally enables designers to minimize the ViP modeling effort. The overall flow of our technique is illustrated in Fig. 7. Three rectangular boxes represent the state-of-the-art EDA technologies that we employ: RTL to ViP translator, transaction-level simulation platform, and simulation based RTL power estimator. We will elaborate them in the following sections.

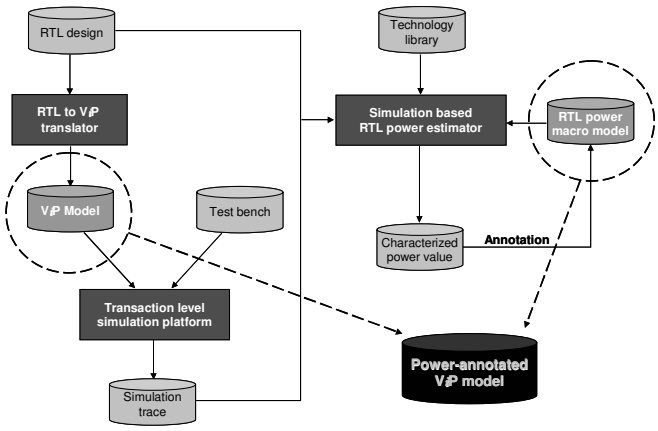


Fig. 7. Overall custom IP power modeling flow.

### A. *ViP model and RTL power macro model generation*

Two intermediate outputs of our flow are circled in dotted circles. The ViP model is generated from RTL description

by using an RTL to ViP translator, and the RTL power macro model is built by extracting representative state machine variables from the RTL source code. At the end of the flow, two intermediate outputs are merged to make the power back-annotated ViP model.

### B. System-level simulation

System-level simulation is performed to obtain simulation traces that are required for the power characterization step. Since the generated ViP model contains all the structural information of the original RTL design, it is possible to dump RTL switching activities to output VCD files. This enables the simulation based power estimation. The inputs to this step are the generated ViP model and testbench. Since most of the IP blocks have a general interface such as a standard bus protocol, the testbench can usually be reused.

### C. Simulation based RTL power estimation

As mentioned previously, we use an RTL power estimator for characterization. The inputs to this step are as follows: the simulation trace in VCD format, the original RTL design, the technology library, and the initial RTL power macro model. The RTL power estimator reads in the technology library file and quickly synthesizes the original RTL design into a gate-level netlist in the target technology. The simulation trace is applied to this gate-level netlist to perform gate-level power estimation. As a result, the initial power macro model is back-annotated with characterized power numbers. The initial RTL power macro model must be evaluated at this point to confirm that it satisfies the accuracy constraint. If the accuracy of the initial model is too low, the RTL power macro model should be refined and this step must be repeated.

This step can be easily substituted with a gate-level estimation tool if the gate-level netlist and testbenches are available. That is, we can come back to this point after the RTL freeze for more accurate power estimation.

### D. Integration of the power model into the ViP model

The last step of our flow is the integration of the power macro model with the ViP model. Since the power macro model is built by extracting representative state machine variables, it is easy to automatically embed the power macro model into the ViP model.

We implement a monitor inside the ViP model, which tracks the list of signals representing the power macro model. Then the power consumption of the IP is reported according to the signal values during simulation. That is, the power back-annotated ViP model can estimate the power consumption on a cycle-by-cycle basis by monitoring its internal switching activities.

### E. Validation of the flow

The validation of our custom IP power modeling technique is in progress. For an 80K-gate IP block that we have tested so far, the modeling effort in terms of man-month is reduced by an order of magnitude, and the accuracy is 80% as compared to the gate-level estimation.

## VII. Memory Power Models

Various types of memories are used for different purposes. SRAM and SDRAM are two of the most commonly used memories in a SoC system. In our framework, SRAM is easily modeled with energy per read, write, or idle operations. In most cases, power variation caused by the data dependency of each operation can be ignored. For SDRAM, we adopt the widely accepted approach, which is well-described in [18]. Read, write, activate, precharge, and refresh power are separately calculated using the DC characteristic numbers provided in the SDRAM datasheets. Our experience shows that these models are sufficient to explore the power consumption in the system context.

## VIII. PowerViP: A SoC Power Estimation Framework

The separately prepared power models are integrated in the ViP framework. In this study, we only present the results for the processor and AXI bus fabrics since the validation of the custom IP power modeling flow is in progress We build a system composed of an ARM926 processor, an AHB-to-AXI converter, AXI bus fabric, a memory controller, and an external SDRAM. A set of test bench suite is run on the ARM processor; we believe that this configuration conforms to our goal of providing a feasibility of our study.

PowerViP produces profile information in HTML format for all the functions run on the ARM926EJ-S processor. As shown in Fig. 8, power consumption and other statistics for each function are reported, e.g., total cycles and bus utilization. Note that the power numbers in Fig. 8 are normalized with respect to the highest power consuming function. It is sorted in decreasing order of total cycles as shown in the second column. From the result, designers can find which function is most power-hungry and start cycle-by-cycle power and performance simulation at the beginning of the function.

Generated by ARMMonia(ARM-core MONitoring Accessory), created by CAE Center Samsung Electronics

Top Name : ARM926_AXI

Core Name : arm926ej-s

Core Type : 926

[Functions Net] -- General

| Function Name | Total Cycles | Instructions | Core Cycles | Cycle/Instr | Core Stall(%) | IBus Active(%) | DBus Active(%) | Total Power | Core Power | I/D$ Power |
|---|---|---|---|---|---|---|---|---|---|---|
| main | 9411 | 1054 | 4873 | 8.929 | 48.220 | 0.361 | 11.136 | 67.618 | 62.704 | 4.916 |
| +BASE_ENTRY+ | 2449 | 135 | 228 | 18.141 | 90.690 | 55.860 | 18.947 | 17.784 | 16.924 | 0.860 |
| dhryStone | 1171 | 456 | 786 | 2.568 | 32.878 | 19.898 | 3.074 | 76.190 | 67.218 | 8.972 |
| strcpy | 866 | 536 | 776 | 1.616 | 10.393 | 4.042 | 17.321 | 83.924 | 73.797 | 10.127 |
| Proc_8 | 460 | 200 | 352 | 2.300 | 23.478 | 9.565 | 16.087 | 77.282 | 69.940 | 7.342 |
| Proc_1 | 356 | 136 | 244 | 2.618 | 31.461 | 17.978 | 4.494 | 77.391 | 67.607 | 9.784 |
| strcmp | 338 | 220 | 268 | 1.536 | 20.710 | 11.243 | 7.101 | 80.662 | 70.748 | 9.913 |
| Func_2 | 328 | 116 | 236 | 2.828 | 28.040 | 15.244 | 3.040 | 77.400 | 68.960 | 8.520 |
| __Heap_Alloc | 264 | 132 | 180 | 2.000 | 31.818 | 19.318 | 27.273 | 75.570 | 66.949 | 8.621 |
| __rt_memcpy_w | 228 | 76 | 164 | 3.000 | 28.070 | 13.596 | 18.860 | 80.505 | 68.598 | 11.907 |
| __rt_entry | 222 | 42 | 64 | 5.286 | 71.171 | 46.396 | 15.315 | 62.567 | 56.004 | 6.563 |
| __rt_lib_init | 208 | 60 | 90 | 3.467 | 56.731 | 53.365 | 23.558 | 67.217 | 59.942 | 7.274 |
| Proc_6 | 204 | 64 | 132 | 3.188 | 35.294 | 26.961 | 3.922 | 74.822 | 66.487 | 8.335 |

Fig. 8. Reported profile in HTML format.

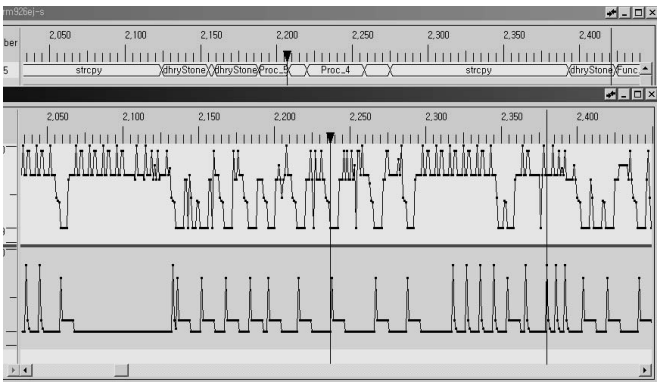


Fig. 9. Graphical power profile in time axis.

Fig. 9 shows the power consumption of the ARM926EJ-S processor and the AXI bus fabrics graphically (on the lower window) as a series of functions such as *strcpy* and *dhryStone* are executed (on the upper window). The power graph shows distinct pattern for each function. In this way, the exact cycle when a power surge occurs can be pin-pointed. Then the debugger can trace the source code line-by-line from the pin-pointed cycle to identify the cause of the power surge. For example, if high instruction cache power consumption is attributed to the power surge, it can be monitored if the instruction cache misses keep occurring at the time, or if the tight loop body incurs frequent branches, causing power-hungry non-sequential instruction cache accesses. All the necessary information is reported in the simulator in a synchronized fashion.

## IX. Conclusions

We tackle the heterogeneity of SoC by adopting different power modeling techniques for different components. Simple power models are used for components that have simple power characteristics, i.e., SRAM and processor core. We build an activity-based coarse-grain power model for the cache memory and develop a component-based approach for bus fabrics. For custom IP blocks, we use RTL power macro modeling combined with the RTL to ViP model translation technique. On the average, our power models for ARM926EJ-S and AXI bus fabrics show 93% and 95% of estimation accuracy respectively as compared to gate-level estimation.

Although our custom IP power modeling strategy still needs to be validated, we are confident that with the help of PowerViP, system designers can explore various architectural choices at the early design phase to find a power-optimal design before the RTL freeze. Currently, our design teams are adopting the framework at the early design exploration stage. We plan to enrich the power model database by providing power models for other commonly used processor core families, bus fabrics, custom IP blocks, and memory devices.

## References

[1] M. Lajolo, A. Raghunathan, S. Dey, and L. Lavagno, "Cosimulation-based power estimation for system-on-chip design," *IEEE Trans. on VLSI Systems*, vol. 10, no. 3, pp. 253-266, 2002.

[2] Samsung Electronics, "Samsung's ViP design methodology reduces SoC design time up to 40 percent," http://www.samsung.com/Products/Semiconductor/News/SystemLSI/SystemLSI_20040914_0000069677.htm.

[3] N. Bansal, K. Lahiri, A. Raghunathan, and S. T. Chakradhar, "Power Monitors: a framework for system-level power estimation using heterogeneous power models," in *Proc. Int. Conf. on VLSI Design*, 2005, pp. 579-585.

[4] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: a first step towards software power minimization," *IEEE Trans. on VLSI systems*, vol. 2, no. 4, pp. 437-445, 1994.

[5] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: a framework for architectural-level power analysis and optimizations," in *Proc. ISCA*, 2000, pp. 83-94.

[6] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The design and use of SimplePower: a cycle-accurate energy estimation tool," in *Proc. DAC*, 2000, pp. 340-345.

[7] G. Gontreras, M. Martonosi, J. Peng, R. Ju, and G.-Y. Lueh, "XTREM: a power simulator for the Intel XScale® core," in *Proc. LCTES*, 2004, pp. 115-125.

[8] A. Sinha and A. Chandrakasan, "JouleTrack: a web based tool for software energy profiling," in *Proc. DAC*, 2001, pp. 220-225.

[9] J. Y. Chen, W. B. Jone, J. S. Wang, H.-I. Lu, and T. F. Chen, "Segmented bus design for low power," *IEEE Trans. on VLSI systems*, vol. 7, no. 1, pp. 25-29, 1999.

[10] C.-T. Hsieh and M. Pedram, "Architectural power optimization by bus splitting," *IEEE Trans. Computer Aided Design*, vol. 21, no. 4, pp. 408-414, 2002.

[11] P. P. Sotiriadis and A. P. Chandrakasan, "A bus energy model for deep submicron technology," *IEEE Trans. on VLSI systems*, vol. 10, no. 3, pp. 341-350, 2002.

[12] L. Benini, A. Macii, M. Poncino, and R. Scarsi, "Architecture and synthesis algorithm for power-efficient bus interfaces," *IEEE Trans. Computer Aided Design*, vol. 19, no. 9, pp. 969-980, 2000.

[13] M. Caldari et al., "System-level power analysis methodology applied to the AMBA bus," in *Proc. DATE*, 2003, pp. 32-37.

[14] U. Neffe et al., "Energy estimation based on hierarchical bus models for power-aware smart card," in *Proc. DATE*, 2004, pp. 300-305.

[15] A. Bona, V. Zaccaria, and R. Zafalon, "System level power modeling and simulation of high-end industrial network-on-chip," in *Proc. DATE*, 2004, pp. 318-323.

[16] K. Lahiri and A. Raghunathan, "Power analysis of system-level on-chip communication architectures," in *Proc. CODES+ISSS*, 2004, pp. 236-241.

[17] S. Ravi, A. Raghunathan, and S. Chakradhar, "Efficient RTL power estimation for large designs," in *Proc. Int. Conf. on VLSI Design*, 2003, pp. 431-439.

[18] Micron Technology, "Calculating DDR memory system power" http://www.micron.com/products/dram/ddrsdram/technote.html,

[19] R. Kumar, V. Zyuban, and D. Tullsen, "Interconnections in multi-core architectures: understanding mechanisms, overheads and scaling", in *Proc. ISCA*, 2005, pp. 408-419.