A Multi-Granularity Power Modeling Methodology for Embedded Processors

Young-Hwan Park, *Member, IEEE*, Sudeep Pasricha, *Member, IEEE*, Fadi J Kurdahi, *Fellow, IEEE*, and Nikil Dutt, *Fellow, IEEE*

***Abstract*—With power becoming a major constraint for multiprocessor embedded systems, it is becoming important for designers to characterize and model processor power dissipation. It is critical for these processor power models to be useable across various modeling abstractions in an electronic system level (ESL) design flow, to guide early design decisions. In this paper, we propose a unified processor power modeling methodology for the creation of power models at multiple granularity levels that can be quickly mapped to an ESL design flow. Our experimental results based on applying the proposed methodology on the OpenRISC and MIPS processors demonstrate the usefulness of having multiple power models. The generated models range from very high-level two-state and architectural/instruction set simulator models that can be used in transaction level models, to extremely detailed cycle-accurate models that enable early exploration of power optimization techniques. These models offer a designer tremendous flexibility to trade off estimation accuracy with estimation/simulation effort.**

***Index Terms*—Digital systems, embedded processor performance, embedded processor power estimation, multi-granularity levels.**

I. Introduction

REDUCING power dissipation is a critical design goal for electronic devices ranging from handheld systems with limited battery capacity to large computer workstations that dissipate huge amounts of power and require costly cooling mechanisms. Designers today must evaluate various power optimizations as early as possible in an electronic system level (ESL) design flow, since design changes are easier and have the greatest impact on application power dissipation at the system level [2], [3]. In order to explore these optimizations, accurate power estimation models are necessary. These models are especially important for chip multiprocessor (CMP) systems with tens to hundreds of relatively simple processors integrated on a single chip. Even a slight inaccuracy in power estimation for a single processor can result in a large absolute error for the chip.

Manuscript received June 05, 2009; revised November 04, 2009. First published January 29, 2010; current version published March 23, 2011. This research was supported in part by grants from SRC (2005-HJ-1330 and 1617.001) and NSF (CCF-0702797).

Y. Park is with the Samsung Advanced Institute of Technology, Yongin-si, Gyeonggi-do 446-712, Korea (e-mail: yh97.park@samsung.com).

S. Pasricha is with the Department of Electrical and Computer Engineering and Department of Computer Science, Colorado State University, Fort Collins, CO 80523-1373 USA (e-mail: sudeep@colostate.edu).

F. J. Kurdahi and N. Dutt are with the Department of Electrical Engineering and Computer Science, University of California, Irvine, CA 92617-3425 USA (e-mail: kurdahi@uci.edu; dutt@uci.edu).

Digital Object Identifier 10.1109/TVLSI.2009.2039153

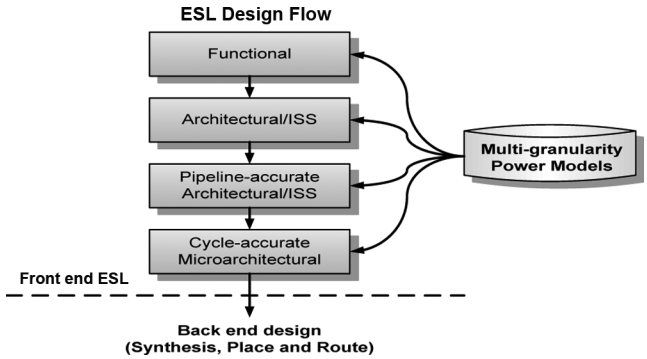


Fig. 1. ESL design flow for embedded processors.

Several system level power estimation approaches have been proposed in recent years focusing on the various components of CMP designs, such as processors [4], [5], memories [6], interconnection fabrics [7], and custom application-specific integrated-circuit (ASIC) blocks [8]. Because of the heterogeneity of these components, power estimation models are usually customized for each component to achieve desired estimation accuracy. In addition, each type of component requires several power estimation models that can be incorporated at the most coarse grain, high levels of abstraction, as well as at the most detailed, low level simulation abstractions.

Fig. 1 shows the typical ESL design stages for embedded processors. The *functional* stage consists of a high level model of the processor that captures its basic functionality. This model is refined down to the *architectural* level, which has a well defined instruction set architecture (ISA). A simulator that captures the ISA at this level is commonly referred to as an instruction set simulator (ISS). In the subsequent stage, the pipeline of the processor is modeled, to create a more detailed *pipeline-accurate architectural* model. This model can simulate instructions flowing through a pipeline, and captures the performance benefits of pipelining, as well as the slowdown due to pipeline stalls. Finally, this model is refined down to the *cycle accurate micro-architectural* level, by adding details of the functional units (data path) and the pipeline (control path), to capture the behavior of the processor at a highly detailed cycle-accurate granularity.

To guide design decisions that affect power dissipation, designers need power estimation models at each of these levels. Existing processor power estimation techniques create power models that map onto and are useful only at a particular level. For instance, the commonly used instruction level power estimation technique [9] assigns a power number to each instruction in a processor ISA. This technique can only be used at an

ESL level that captures the ISA. Thus, while this technique is readily applicable at the architectural level, it cannot be easily used at the higher functional level which is unaware of the ISA. Furthermore, if this technique is used at the lower levels, it fails to exploit the additional accuracy in the control and data paths and suffers from an abstraction mismatch. Similarly, cycle accurate power estimation tools such as Wattch [4] and SimplePower [5] are applicable to the detailed micro-architectural level of the ESL design flow, but cannot be easily ported to higher level architectural/ISS models that lack micro-architectural detail. The mismatch between power model granularity and level of detail captured at an ESL design level thus limits the applicability of current power estimation techniques across an ESL flow.

In this paper, we propose a comprehensive multigranularity power model generation methodology that spans the entire ESL design flow (see Fig. 1). Using industry-standard design flows, our methodology can quickly generate multiple power models ranging from the simplest two-level, coarse grained model for early power estimation, to the most accurate cycle accurate model that allows designers to explore the impact of using power optimizations with minimal manual interference and effort. Our proposed approach is based on the concept of hierarchical decomposition. This decomposition is aided by a tripartite hyper-graph model of processor power that can be iteratively refined to create power estimation models with better accuracy. The methodology serves a vital function in supplying a designer with multiple derivative processor power estimation models that match the increasing accuracy of the design, as it is successively refined from the functional, to the architectural and then down to the cycle-accurate micro-architectural stages in an ESL design flow. We demonstrate the feasibility of our approach on an OpenRISC and MIPS processor case study, and present results to show how multigranularity power models generated for the processors provide designers with the flexibility to tradeoff estimation accuracy and simulation effort during system-level exploration. Our approach targets RISC type processors which are often used in large CMP or networks-on-chip (NoC) [31].

In addition to generating multigranularity processor power models, an important benefit of our methodology is to quickly create detailed cycle accurate processor power estimation models that can allow designers to explore the impact of using fine grained power saving techniques at the system level, as well as comprehensively explore the impact of ISA extensions (i.e., adding new custom instructions). The cycle accurate power estimation enabled by our methodology can bring about many additional benefits. First, we can identify the exact time that the processor uses larger power due to a specific program routine. This can be vital for achieving power savings. If we know which function routines cause high power consumption, we can replace these functions with less power consuming algorithms or an optimized program [26]. Second, cycle accurate power estimation is useful for determining peak power, which allows designers to estimate the thermal and electrical limits of ICs and the appropriate packaging for them [29]. In particular, for CMPs, simply accumulating average power of each processor can result in too optimistic or pessimistic peak power estimates which are far from the actual values. Therefore it is important to know cycle accurate power data for each component to compute a more realistic peak power for the entire system. Finally, cycle accurate power estimation can aid in the exploration of counter-defense strategies to prevent side-channel attacks that exploit information about the power dissipation, timing and electromagnetic leaks to compromise systems [28].

II. Related Work

Processor power estimation has been the focus of several research efforts over the past several years. One of the simplest processor power estimation models is a two-state model with one of the states representing the processor when it is busy, and the other representing an idle state [2]. A more accurate and popular power estimation technique for processors is instruction level power estimation, which was first proposed by Tiwari *et al.* [9]. Several subsequent research efforts [10]–[16], [26], [27] have applied instruction level power estimation to obtain high level power estimates for various processor variants such as digital signal processors (DSPs) [11], [26], VLIW processors [14], [27], and the Intel XScale [16]. The technique is based on the simple observation that each instruction can be assigned its own power cost (base cost). However for more accurate power estimation, some of these works also include inter-instruction effects in the power model. The cost of a pair of instructions (additional cost) is varied depending on the combination of these instruction pairs. This inter-instruction effect makes the power modeling more complicated. If we consider a CISC processor which has hundreds of instructions, these inter-instruction effect needs nontrivial complex data and corresponding experiments and analysis. To overcome this problem, some studies have tried to cluster instructions based on the similarities of their power cost [26], [27]. However, since these approaches do not consider variable influences at each pipeline stage, they are not sufficient for cycle accurate power modeling. These inter-instruction effects also need to be evaluated for each separate pipeline for cycle accurate power data. More details on this will be described in Section III-D.

While the above techniques are useful for early estimation of average power, they are not very accurate for cycle-level power estimation. For more accurate processor power estimation, structural modeling [4], [5], [17], [32] is a widely used approach, which creates power models for smaller decomposed sub-units in the processor. These power models observe the activity of the units in the processor and use this information for estimating power. Power estimation tools such as Wattch [4] and SimplePower [5] use this approach, and are relatively well known because they can cooperate with the widely used processor simulation platform SimpleScalar [18]. Even though these tools are popular, they have their limitations. Wattch focuses on regular structures such as memory array and CAM structures, for which it is relatively easy to calculate switched capacitance, but not on the complex combinational logic often found in processors. SimplePower relies on a lookup table (LUT) that contains the switch capacitance for each input transition of the processor functional units. This methodology is known to be accurate for small units, but if the input size is large, it is impractical to have all the cases in the table. Techniques

used to overcome this limitation end up sacrificing estimation accuracy. More recently, Bhattacharjee *et al.* proposed a power modeling approach using field-programmable gate array (FPGA)-based emulation [32]. They claimed that they obtained about 35× speed up over pure software-based simulation with the aid of using hardware performance counters (HPCs) in an FPGA-based emulator. However, this methodology needs extra work such as implementation of a processor on FPGA, changes of HDL code for implementing custom counters and modification of OS kernel for reading the HPCs periodically to provide the cycle accurate runtime power profiling, a process which may be error-prone.

All of these processor power estimation techniques have tradeoffs between designer effort, accuracy and simulation speed. To the best of our knowledge, there has not been any approach that can comprehensively generate several consistent power models for a processor to plug into the various stages of an ESL design flow. Our methodology can provide an easy way for designers to create multigranularity models by simply varying certain parameters to tradeoff estimation accuracy and simulation speed. Another important differentiation between this work and prior work is that our methodology generates both average and maximum power estimates, and provides these estimates dynamically (i.e., for a given window of time). This helps in early design space exploration of CMPs with a large number of processors, and facilitates the realistic optimization of such systems. Finally, we note that while our approach concentrates on the processor cores, it is complementary to existing memory power estimation models such as CACTI [6] which provide estimates of caches and other memories peripheral to the processor core and are comparable in terms of accuracy and flexibility to our models.

III. Power Modeling Methodology

In this section we present details of our power modeling methodology. In Section III-A, we present details of a 3-D power contribution lookup table (LUT) that drives our overall power estimation effort. Section III-B describes the concept of a tripartite hyper-graph to represent a processor, which can be iteratively decomposed to develop multigranularity processor power estimation models for various ESL design stages. Section III-C presents multigranularity power models generated by our approach for an ESL design flow. Section III-D presents details on how to customize power modeling for different processors, to trade off estimation effort, speed and accuracy. Finally, Section III-E presents an overview of our power model generation flow.

A. 3-D Power Contribution LUT

During program execution, an instruction in a processor ISA activates different functional sub-units, and consequently dissipates varying amounts of power as it traverses the processor pipeline stages. To accurately characterize the power contribution of an instruction, we create a 3-D LUT, as shown in Fig. 2. Assuming K processor functional units, M pipeline stages, and N instructions in the ISA, we create a table that holds the power dissipation for each instruction, at each pipeline stage, for every functional unit in the processor. The power

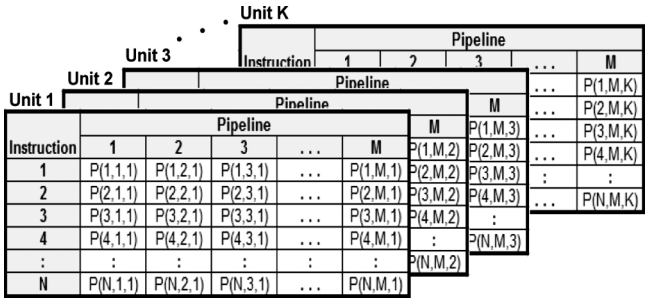


Fig. 2. 3-D power contribution LUT.

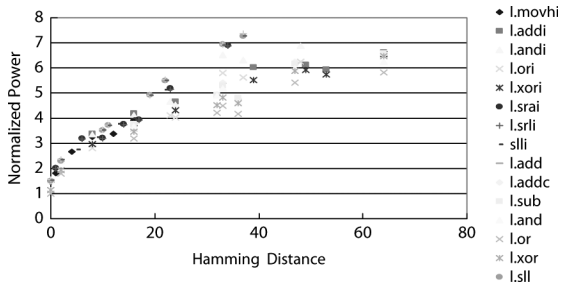


Fig. 3. Relation between HD and power of several instructions at EX stage in ALU unit of the OpenRISC processor.

data stored in the LUT represents pure switching power due to instructions traversing through pipeline stages. The total power is obtained after addition of an offset power component (i.e., NOP power) to the switching power for all pipeline stages. For improved accuracy, we create a set of three such 3-D LUTs, corresponding to average power, minimum power and maximum power. For 2-operand instructions, these power ranges can be obtained by varying the data operand values from minimum to maximum (e.g., 0x0000 to 0xFFFF) which will vary the Hamming Distance (HD) between the two operands. A larger HD is found to result in greater power dissipation [8].

Fig. 3 shows the normalized power for several instructions from the OpenRISC ISA [19], for varying HD values of operand data. All power data is normalized to the minimum power value of the *l.or* instruction. It can be seen that power dissipation between the minimum and maximum HD cases varies by as much as 6×, justifying the need for three 3-D LUTs. Linear interpolation is used to estimate the power entries in the LUTs for arbitrary data values using their HD. Section III-E describes how we populate the 3-D LUTs in more detail, and how inter-instruction and other unpredictable power dissipation factors are captured using regression compensation.

B. Processor Tripartite Hyper-Graph

For the purpose of creating multigranularity processor power models, we can conceptually represent a processor as a tripartite hyper-graph $H(P) = \langle V, E \rangle$, as shown in Fig. 4(d). The set of vertices V is partitioned into three disjoint sets: $I = \{i_1, i_2, \ldots, i_N\}$, $S = \{s_1, s_2, \ldots, s_M\}$, and $U = \{u_1, u_2, \ldots, u_K\}$ corresponding to the set of processor instructions, pipeline stages, and functional units, respectively. Each dimension (entry) of the 3-D LUT represents exactly one of these disjoint sets of the graph and thus the graph can

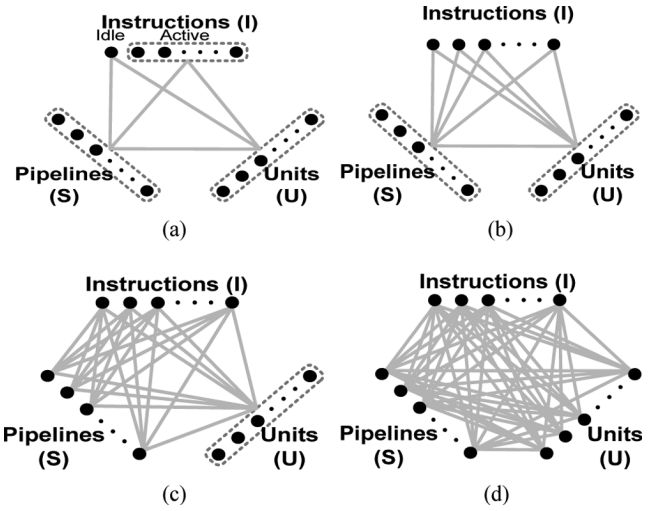


Fig. 4. Tripartite hyper-graph H(P): (a) simplest two-state power model (*Level 0*); (b) power model with set I decomposed (*Level 1*); (c) power model with sets I, S decomposed (*Level 2*); (d) power model with sets I, S, U decomposed (*Level 3*).

be regarded as a graphical representation of the LUT. The ternary edges (or hyper-edges) E is a set of triples (i, s, u), $i \in I$, $s \in S$, $u \in U$ that represent the ternary (i.e., three-way) association between an instruction, its power dissipation across the pipeline stages it traverses, and the functional units it activates. The weights of those hyper-edges are populated from the 3-D LUTs described in the previous subsection. A reduced graph, $H'(P)$ can be obtained by clustering disjoint subsets of I, S and U, resulting in $1 \leq |I'| \leq N$, $1 \leq |S'| \leq M$ and $1 \leq |U'| \leq K$ clusters for each of the three sets of vertices. The number of hyperdeges is also reduced as one hyper-edge exists between each triplet of clusters.

The tripartite hyper-graph offers a convenient way to represent the granularity of different processor power models. Consider a simple two -state processor power model, with only two power values: for the active and idle states. Such a model can be represented as shown in Fig. 4(a), with a triple (i, s, u), where $|I'| = 2$ (an idle or NOP, and an active instruction), $|S'| = 1$, and $|U'| = 1$. Such a coarse grain model does not require information about pipeline stages or functional units. More accurate, finer granularity power models can be obtained by decomposing subsets I', S', and U' in the hyper-graph, examples of which are shown in Fig. 4(b)–(d). These models successively represent more accurate models of the power dissipation for multiple instructions I, as they traverse the pipeline stages S, and activate U functional units. Thus, a reduced graph, $H'(P)$ can provide an effective way to represent the coarser grained power model. An example of this is presented in Section III-D.

C. Processor Power Models for an ESL Flow

Having introduced the concept of a tripartite hyper-graph that allows multigranularity power model representation, we now present power models that can be mapped to the various stages of an ESL design flow, shown in Fig. 1. There are the following four power models that map to the appropriate four major ESL stages.

Level 0 (functional): A two-state (active, idle) coarse grained power model is used for the functional stage, and is shown in Fig. 4(a). The simplest processor power model of [2] can be classified under this level.

Level 1 (architectural/ISS): At the architectural level that supports ISA simulation, the instruction subset is decomposed so that each instruction has a power value ($|I'| = N$, where N is number of instructions in ISA), as shown in Fig. 4(b). Since the pipeline stages and functional units are not necessarily modeled at this stage, their effect on power dissipation cannot be modeled, and therefore subsets S' and U' are not decomposed ($|S'| = |U'| = 1$). Instruction level power modeling described in [9]–[16], [26], and [27] can be classified under this level.

Level 2 (pipeline-accurate architectural): When the pipeline is modeled at the pipeline-accurate architectural level, the power model can be further refined by decomposing the pipeline stage subset S' ($|S'| = M$, where M is the number of pipeline stages), as shown in Fig. 4(c). This represents more accurate power dissipation, as it accounts for the effects of pipelined execution, including any stalls and flushes.

Level 3 (cycle-accurate micro-architectural): When the structural units are additionally modeled cycle-accurately, as is the case at the cycle-accurate micro-architectural level, then the power model can be further refined by decomposing the functional unit subset U' ($|U'| = K$, where K is the number of processor function units), as shown in Fig. 4(d). This is the most accurate power model that gives very reliable cycle-accurate power dissipation information. Structural power modeling described in [4], [5], and [17] can be classified under this level.

Note that the finer grained, lower level power models are extremely accurate, but require significantly greater modeling effort and simulation overhead.

D. Power Model Customization

The multigranularity power models described above map conveniently to the different ESL design stages. However, different processors have different functional (i.e., instruction set), temporal (i.e., pipelining), and structural (i.e., functional unit) complexities. These may require more flexible power models, to achieve the best possible tradeoff between accuracy and simulation effort. This flexibility in our power model generation methodology is achieved by appropriately varying the number of elements in each of the subsets I', S', and U'.

1) Instruction Set Clustering: One possible power model customization is to reduce the number of instructions considered for power estimation, by clustering similar power dissipating instructions into groups. Fig. 5 shows an example of our hyper-graph clustering algorithm to automatically generate different instruction set groupings. Recall that each instruction in the ISA has a power dissipation range, obtained from the minimum, average, and maximum power in the 3-D LUTs. The algorithm attempts to reduce the complexity of an instruction level power model (*Level 1*) by clustering instructions that have similar behavior and power dissipation characteristics. If some instructions span a similar power range within a deviation threshold (T_d), we can cluster them together. For example, as shown in Fig. 5, if the difference in power between instruction $I1$ and

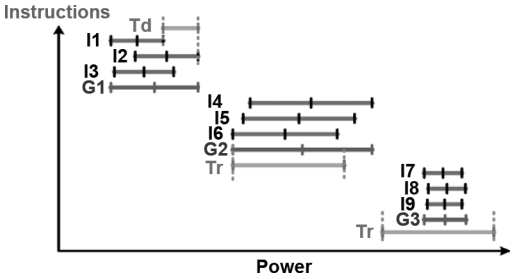


Fig. 5. Instructions and clustering groups.

TRIPARTITE HYPERGRAPH DEVIATION CLUSTERING

```
1 Td ← deviation threshold value
2 for u ← 1 to K do
3   c ← 0
4   for i ← 1 to N do
5     if P[i,u] does not belong to G then
6       create G[c,u]
7       initialize G[c,u] with P[i,u]
8       LUT[i,u] ← c
9       for n ← i+1 to N do
10        if P[n,u] does not belong to G then
11          for s ← 1 to M do
12            check deviation (P[n,s,u], P[i,s,u]) ≤ Td
13          if all deviation ≤ Td then
14            update G[c,u] with P[n,u]
15            LUT[n,u] ← c
16      c = c + 1
17 return G
```

Fig. 6. Tripartite hyper-graph clustering algorithm with T_d.

$I2$ is smaller than the threshold value, we can regard these instructions as similar enough to be clustered together. And the difference is evaluated by the equation

$$|P\max(I1) - P\max(I2)| < T_d$$
$$|P\min(I1) - P\min(I2)| < T_d. \quad (1)$$

A larger threshold value will allow more instructions to be grouped together. In the figure instructions *I1–I3* constitute group $G1$, while *I4–I6* are part of group $G2$, etc., using this method.

Fig. 6 outlines the pseudo-code for the tripartite hyper-graph used in the clustering algorithm with deviation threshold T_d. Initially the T_d value is set at line 1. A loop for the K units (line 2) builds separate clusters, one for each unit. The total number of instruction clusters, c is initialized to zero (line 3). Then, for each instruction i, we check whether it already belongs to an instruction cluster (group). If it does not, a new group $G[c,u]$ is created and the power value of $P[i,u]$ is used to initialize the new group G. At the same time we create a LUT for referencing group power value instead of instruction power value for future use (lines 4–8). Next, instruction i is compared to all the remaining instructions. If there is an instruction n which does not yet belong to any cluster, its power $P[n,s,u]$ is compared with the power of instruction i, $P[i,s,u]$ for all pipeline stages to check if (1) is satisfied (lines 9–13). If (1) is satisfied for all the pipeline stages, the power value for the instruction is used to update the group $G[c,u]$. Now, $G[c,u]$ will have the new max, min, and average power after adding the new $P[n,u]$ value (line 14). The largest max value in the group is used for the new max for the group and the smallest min value is also set with the same method. The LUT is also updated for future referencing of the clustering group and corresponding instruction (line 15), and the total number of instruction clusters, c is incremented (line 16). Finally, the clustered groups G are returned (line 17).

TRIPARTITE HYPERGRAPH RANGE CLUSTERING

```
1 Tr ← range threshold value
2 for u ← 1 to K do
3   retreive cnt_clust from G[u]
4   for c ← 0 to cnt_clust – 1 do
5     for s ← 1 to M do
6       clust_range = G[c,s,u]->max_pow - G[c,s,u]->min_pow
7       if (clust_range ≤ Tr) then
8         G[c,s,u]->keep_all_flag = 0
9         discard Min and Max power value of G[c,s,u]
10      else G[c,s,u]->keep_all_flag = 1
11 return G
```

Fig. 7. Tripartite hyper-graph clustering algorithm with T_r.

An approach to further improve power estimation speed uses a range threshold (T_r) to discard the min and max values for an instruction group, if the power range for the group is smaller than T_r. The goal is to reduce the number of LUT entries for an instruction (group) if there is minimal variation between its min and max values. In Fig. 5 for instance, $T_r > P_{\max}(G3) - P_{\min}(G3)$, and therefore for $G3$, min and max values are discarded, and we just keep the average value. For group $G2$, $Tr < P_{\max}(G2) - P_{\min}(G2)$, and so we do not discard its min and max values, which would cause a more significant impact on estimation accuracy. Both of these threshold based approaches allow faster power estimation from *Level 1* onwards (by reducing the instruction space and pruning LUTs) at the cost of a slight inaccuracy.

The pseudo code for clustering using the range threshold T_r is shown in Fig. 7. The initial range T_r is set to the user-defined value (line 1). For all K units, the total number of clusters (cnt_clust) is retrieved for each unit u (lines 2–3). Then for all clusters c and pipeline stages s, the range of the cluster $G[c,s,u]$ is computed (lines 4–6). If the range is smaller than T_r we set the *keep_all_flag* to 0 and discard the minimum and maximum power values of that cluster (lines 7–9). Otherwise we set the *keep_all_flag* to 1 and retain all the power values including average power value (line 10). Finally the simplified cluster G is returned.

Clearly, changing the values of T_d and T_r results in different clusterings, and therefore different power models. Increasing T_d has a major impact on reducing LUT size, as shown in Fig. 8(a). Larger T_d values reduce the amount of data to be stored in the LUT exponentially while varying the T_r value has relatively less influence on reducing LUT size. This is because T_r only helps to reduce the Min. and Max. power values, whereas T_d groups several instructions together, which is a more effective way to simplify the LUT. Even though the influence of T_r is relatively smaller, we can see its linear influence on reducing the LUT size in the figure. The deviation value σ is computed as

$$\sigma = \sqrt{\frac{1}{N \times M} \sum_{i=1}^{N} \sum_{j=1}^{M} (p_{ij} - \bar{p}_{ij})^2} \quad (2)$$

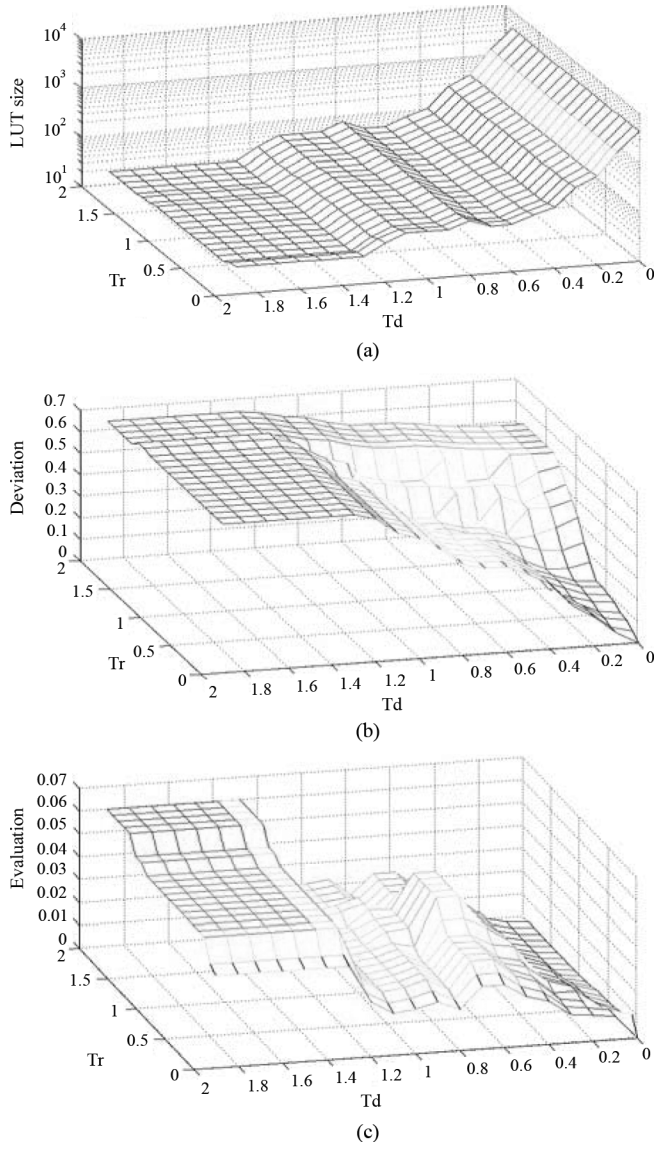


Fig. 8. Impact of varying T_d and T_r values: (a) LUT size; (b) deviation; (c) evaluation.

where N is the number of total instructions, M is the number of total pipeline stages, P_{ij} is power value of cluster i at a pipeline j, and $\bar{P}_{ij}$ is the original power value of an instruction i at a pipeline j.

Deviation values for different T_d and T_r values are shown in Fig. 8(b). We note that the deviation is increased when either the T_d or T_r value is raised starting from zero deviation for both T_d and T_r. These power values including T_d and T_r values are normalized to the power value of idle status which is 1. From the point when $T_d = 1.2$, the deviation is saturated and it is almost close to the worst deviation compared to the original value.

In order to determine the optimum T_d and T_r values which can give the best accuracy with a given value of LUT size, we propose a simple scoring function

$$\text{Score} = \frac{1}{\sigma \times C} \tag{3}$$

where σ is the deviation and C is the number of components in the LUT (representing its size). The maximum deviation, $\max\_\sigma$ is a constraint and so the goal of the scoring function is finding the $\min\{C(T_d, T_r)\}$ and $\min\{\sigma(T_d, T_r)\}$ subject to $\sigma(T_d, T_r) < \max\_\sigma$. We found that selecting $T_d = 0.6 \sim 0.7$ and $T_r = 0.8 \sim 1.2$ can give us relatively good power data with 40% of the max. σ and a given size of LUT, as shown in Fig. 8(c). Note that these values are relevant only for RISC-type processors evaluated in this work. The threshold values will likely be different for other types of processors.

An example of instruction clustering is illustrated in Fig. 9. If we assume that we allow more strict 20% max. deviation which is shown in Fig. 8(b), we can get the highest evaluation value when $T_d = 0.3$ and $T_r = 0.4$ from Fig. 8(c). The clustering shown in Fig. 9 is obtained with these T_d and T_r values. These power values shown in the figure are normalized to the NOP instruction power and power ranges of instructions at the execute (EX) pipeline stages. The range of representative clusters in each group is also shown in the first of the clustering groups. We verified that with our tripartite hyper-graph clustering approach, we can cluster instructions without the overhead of manual instruction categorization based on functional behavior, as done in [26]. We can see large variations in power depending on the instruction types with our approach. Furthermore, our technique is more accurate than the mean clustering algorithm [27] which only considers mean values to cluster instructions that have similar mean power; the mean clustering algorithm results in large computational error especially for cycle accurate power data because it has more dynamic ranges at each decomposed pipeline stage. We also observe from Fig. 9 that clusters C0, C3, C8, and C9 discard the Min and Max power values and thus do not require complicated interpolation (unlike other cluster groups), which saves LUT storage and computation time.

Fig. 10 shows a simplified conceptual diagram of the tripartite hyper-graph $H'(P)$ after executing this instruction set clustering algorithm. As shown in Fig. 10, since the number of instructions in set I is now reduced to I' (i.e., number of clusters), the number of hyper-edges connecting each set (I', S and U) is reduced significantly as shown in Fig. 10. The clustering is an extremely effective way to save the LUT size.

2) Further Decomposition and Inter-Instruction Effect: It is possible that greater accuracy is required than the accuracy supported by the models described in Section III-C. In such a case, we can improve accuracy by using two strategies.

First, we can further decompose pipeline and/or functional units, and consider their power contribution separately. For instance, the EX pipeline stage can be further decomposed into multiple stages, or a register file unit can be decomposed into several sub-banks. In our approach, we use a ranking scheme that iteratively performs decomposition starting with the unit with the highest rank, which coincides with the largest power dissipation magnitude and variation.

Second, we can also perform a regression compensation step to improve estimation accuracy. In this step, we account for hard to determine factors that contribute to power dissipation, including complex inter-instruction influences due to multiple instructions simultaneously traversing the pipeline. Traditional instruction-level power modeling for a single issue processor has typically only considered inter-instruction effects (additional cost) of neighboring instructions, resulting in a

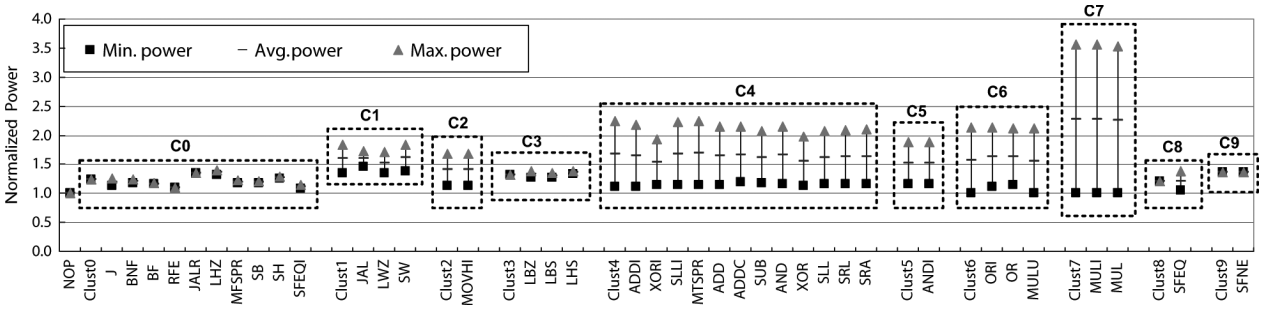


Fig. 9. Normalized power value of example of clustered instructions.

complexity of $O(N^2)$ where N represents the total number of the instructions in the instruction set. For cycle accurate power modeling, the problem is more complex and we have to include variations caused by instructions that are in different pipeline stages. Thus the complexity of the inter-instruction effect is now bounded by $O(M*N^2)$, where M is total number of pipeline stages in the processor. Note that the regression adjustment is relevant only for lower levels in the ESL flow that capture structural and pipeline details (e.g., *Level 3*). The cycle power for the model with regression adjustment [20], based on testbench simulation and subsequent curve fitting can be expressed as

$$P_{\text{Cycle}} = \sum_{s=1}^{M} (P(I_s) + \alpha(I_s\,|I_{s-1})) \tag{4}$$

where M is the number of pipeline stages in the processor, $P(I_s)$ is the base power cost of an instruction at a pipeline stage s, and $\alpha(I_s|I_{s-1})$ is the additional power cost due to the instruction sequence $(I_{s-1} \rightarrow I_s)$ at consecutive pipeline stages. Our clustering algorithm (see Section III-D) can significantly reduce the complexity of this process to $O(M * |C|^2)$, where C is the number of clusters (instead of total number of instructions). Therefore, the simplified equation now becomes

$$P_{\text{Cycle}} = \sum_{s=1}^{M} (P(C_s) + \alpha(C_s\,|C_{s-1})) \tag{5}$$

where $P(C_s)$ is the base power cost of a cluster at a pipeline stage s and $\alpha(C_s|C_{s-1})$ is the additional power cost due to the cluster sequence $(C_{s-1} \rightarrow C_s)$ at consecutive pipeline stages. Our experimental results in Section IV show that such a compensation step can improve estimation accuracy noticeably.

E. Power Model Generation Methodology

The overall methodology to build the multigranularity power models for an ESL design flow is shown in Fig. 11. The methodology consists of two major flows: the 3-D power LUT generation, shown on the left and the power model generation for the desired ESL design stage, shown on the right. In Step 1, the processor RTL (Verilog) design is synthesized to a gate-level net-list using Synopsys Design Compiler [21], for the target technology cell library. In Step 2, the gate-level simulation is performed using NC-Verilog [22], with a special purpose tuning testbench shown in Fig. 12. This testbench consists of all the instructions in the ISA separated by appropriate number of

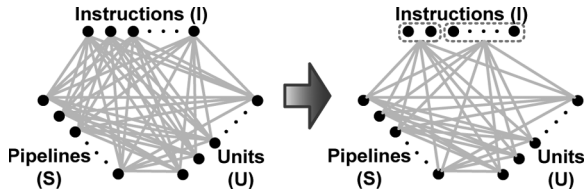


Fig. 10. Simplified tripartite hyper-graph after clustering instructions.

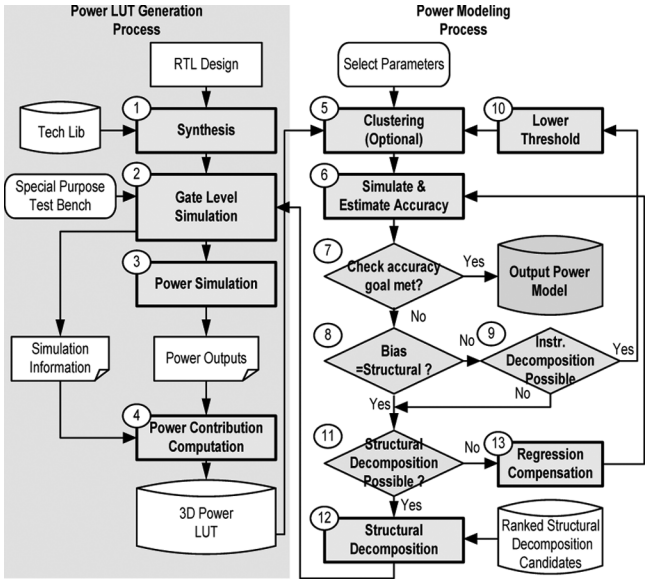


Fig. 11. Power model generation methodology.

NOPs, to isolate the power dissipation for each instruction type, and make sure that that instruction is the only one traversing the processor pipeline. Operand data values with minimum, average and maximum Hamming distances are used for each instruction. This step provides us with simulation information such as timing, control signals that indicate accessed functional units, and information about the activity of instructions in each pipeline stage. In Step 3, power simulation is performed using the PrimeTime PX tool [21], to generate gate level power data, which is decomposed for each functional unit using Perl scripts. The generated information in Steps 2–3 is provided to Step 4, which generates the 3-D LUTs.

The generated 3-D LUTs are then provided to the power modeling flow. Depending on the granularity of the model required (e.g., *Level 0*,*1*, *2*, or *3*, Section III-C), the appropriate values to

```
Instr1 (min HD)
NOP
…
NOP
Instr1 (average HD)
NOP
…
NOP
Instr1 (max HD)
NOP
…
NOP
Instr2 (min HD)
NOP
…
```

Fig. 12. Special purpose testbench program.

create the tripartite hyper-graph are specified. The target accuracy goal for the power model is also specified. Additionally, if a tradeoff between accuracy and simulation effort is desired, then the user should provide deviation and range threshold values, rank values for the pipeline/functional units to guide structural decomposition, and a bias value to indicate preference for instruction set or structural decomposition, if accuracy goals are not met. In Step 5, an optional instruction clustering-based optimization is performed (see Section III-D). In Step 6, the power model is integrated into an ESL stage and simulated to obtain power information for the tuning benchmark, and compared with gate-level simulation data for the same benchmark. If the power model meets the accuracy goal (Step 7) then it becomes the output power model, for use in ESL power simulation with any application. If the accuracy goal is not met, then we need to refine the power model. The bias value is checked to determine if instruction set decomposition or structural decomposition is favored. Either decomposition increases the sizes of one or more of the sets in the hyper-graph, and improves accuracy, at the cost of estimation/simulation effort. If the bias value in Step 8 favors instruction set decomposition, then we check if such decomposition is possible (i.e., check if instruction groups exist that can be decomposed). If decomposition is possible (Step 9), we reduce the threshold values (Step 10) and go back to Step 5 and redo the clustering, which will then create fewer (or no) groups. Otherwise we proceed to the structural decomposition. Provided it is possible for at least one of the ranked components to be decomposed (Step 11), we decompose the highest ranked unit (Step 12), and go back to Step 2, as shown. If no decomposition is possible, then we apply regression compensation in Step 13 (see Section III-D) and repeat the flow from Step 6 onwards, till the output power model with appropriate accuracy is generated.

IV. Processor Power Modeling

To evaluate the effectiveness of our multigranularity power model generation methodology, we use the freely downloadable open source OpenRISC [19] and MIPS ISA [25] processors as case studies.

A. OpenRISC Processor Power Modeling

OR 1200 (part of the OpenRISC 1000 family [19]) is a 32-bit scalar RISC processor with a Harvard architecture. Fig. 13 shows a basic block diagram of the processor, which has a four-stage pipeline, basic DSP functionality, and virtual memory support with an MMU. The CPU core has 32 general purpose 32-bit register file (RF) implemented as two synchronous dual-port memories. The controller (CTRL) manages instruction decoding and generates proper control signals for each of the pipeline stages for each instruction. The integer execution unit implements 32-bit integer instructions such as arithmetic, compare, logical, and rotate/shift instructions. Most integer instructions can be executed in one cycle. A few instructions such as multiply require more cycles (four-cycle latency). The Multiply/MAC (MULT_MAC) unit executes multiplication and basic DSP MAC operations. The MAC unit is fully pipelined so that it can fetch new MAC operations at every clock cycle. The processor also has special purpose registers, and an exception unit which handles exception cases such as external interrupt request, a system call, or attempting to execute unimplemented opcode, etc. It has been claimed that when the processor is implemented in a typical 180-nm 6LM process, it can provide over 300 Dhrystone, 2.1 MIPS at 300 MHz and 300 DSP MAC 32×32 operations that is at least 20% better than other competitors in this class (32-bit RISC processors) such as ARM10 and Tensilica RISC processors [19].

1) OpenRISC Power Models: Fig. 14 shows the multigranularity power models generated for the OpenRISC processor using our methodology. *Level 0*, *1_b*, *2_b*, and *3_a* correspond to *Levels 0–3* of a typical ESL flow, as described in Section III-C. Three additional levels are also created, customized for the OpenRISC, to further tradeoff accuracy and estimation effort. *Level 1_a* uses instruction clustering to reduce the number of instructions considered in ISS models at *Level 1* of the ESL flow. *Level 2_a* abstracts up the EX stage of the pipeline as a single stage. Finally, *Level 3_b* includes regression compensation on the most detailed model (*Level 3* of the ESL flow) to further improve accuracy over gate-level estimates.

2) Experimental Results: In this section, we present results of applying our power model generation methodology on the OpenRISC processor case study. Accuracy of various system level models is measured by comparing with the data of a gate level simulation as shown in Fig. 15. The power models in Fig. 14 were generated using our proposed methodology in Fig. 11, for the 65-nm TSMC standard technology library implementation of the OpenRISC processor. The models were then incorporated into appropriate simulation models of the OpenRISC in SystemC 2.2 [23] for the four levels of modeling abstraction in an ESL design flow (see Fig. 1). The estimated power obtained from the simulation of multiple testbenches (*dhry, des, mul, tick, cbasic, basic*) at these ESL modeling abstractions was compared with gate level power estimates at the 65-nm node. The absolute power estimation cycle error, for each of the generated multigranularity ESL power models, when compared to gate level power can be calculated as

$$E_{AC}^{(i)} = \frac{\left| P_S^{(i)} - P_G^{(i)} \right|}{P_G^{(i)}} \times 100\% \tag{6}$$

where $E_{AC}(i)$ is absolute cycle error at cycle i, $P_S(i)$ is cycle power obtained from system level simulation with the generated

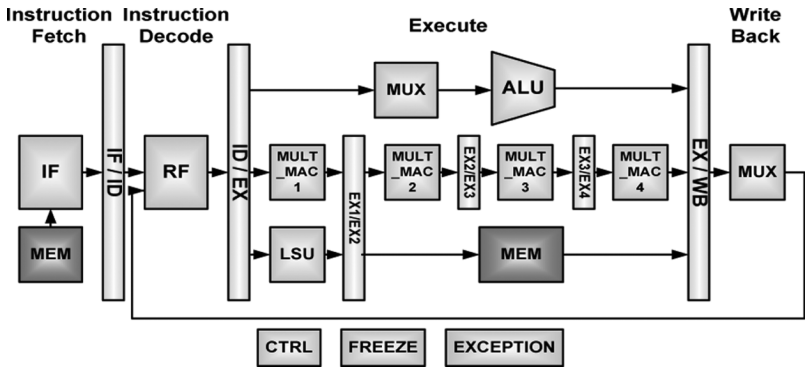


Fig. 13. CPU/DSP core architecture of OR1200.

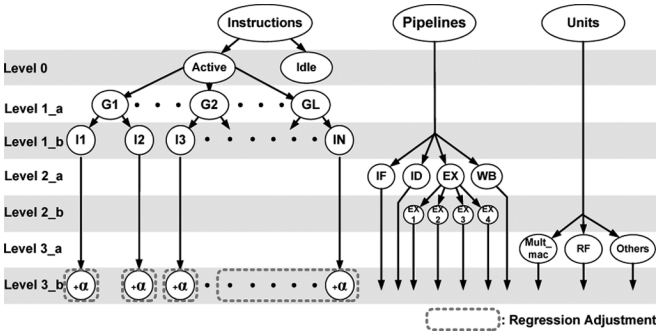


Fig. 14. Hierarchical power model for OpenRISC processor.

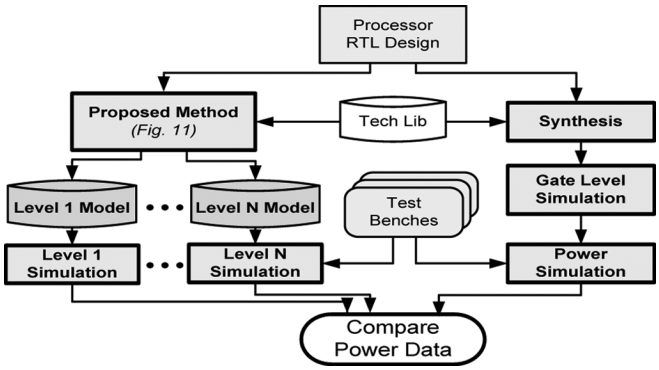


Fig. 15. Measuring accuracy of system level power data with a gate level simulation.

power model, and $P_G(i)$ is cycle power from gate level simulation. Note that while the higher level power models do not have the same clock cycle period as in detailed gate level simulation, for comparison purposes we sample the estimated power from the models at the gate level clock cycle period. The average absolute cycle error can then be formulated as

$$E_{AAC} = \frac{\sum_{i=1}^{N} E_{AC}^{(i)}}{N} \tag{7}$$

where E_{AAC} is the average absolute cycle error and N is the total number of simulation cycles.

Fig. 16 shows the average absolute cycle error (E_{AAC}) and relative estimation effort in terms of simulation overhead, for the generated power models for OpenRISC. The power model

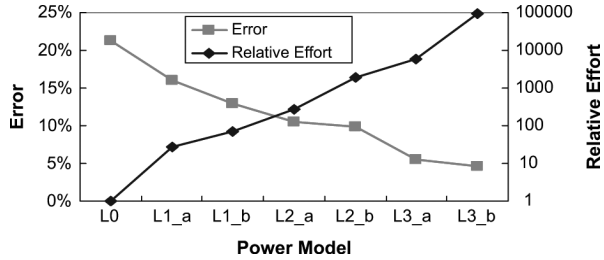


Fig. 16. Average absolute cycle error for power models.

at *Level 0* has a large error of over 20%, which subsequently reduces for the more detailed power models. The *Level 3_b* power model has an approximately 5% error, which is extremely good compared to gate level estimates. The error in such a detailed model occurs because of several factors, such as the inability to capture the layout and consequently accurately model intra-processor interconnect length, and wire switching. As part of our ongoing work, we are integrating our previous work on interconnect power estimation [24] that has the capability to create an early layout and provide routing information at the ESL level for embedded systems, which can improve accuracy. In addition to estimation accuracy, Fig. 16 also compares relative effort, in terms of simulation time required for power estimation at the various levels in the ESL design flow. The variation in simulation time is an artifact of the amount of detail that must be simulated. Higher level models such as *Level 0*, *1_a*, and *1_b* are faster because they do not capture the processor pipeline and structural details in a cycle accurate manner, unlike the lower level models. The generated power models allow a designer to tradeoff power estimation accuracy with "estimation/simulation" overhead. Based on the availability of simulation models, accuracy goals, or desired simulation speed, designers can generate and use the appropriate power model using our methodology.

The results in Fig. 16 are presented for the tuning benchmark (see Section III-E) that was used to create the power models. In order to show that the power models are applicable to any benchmark, we determined cycle power estimates for other benchmarks executed on the OpenRISC processor. Fig. 17 shows the average error and average absolute cycle error for the *Level 3_b* model, compared to gate level estimates, for several benchmarks. We chose *Level 3_b* for the comparison because it is the

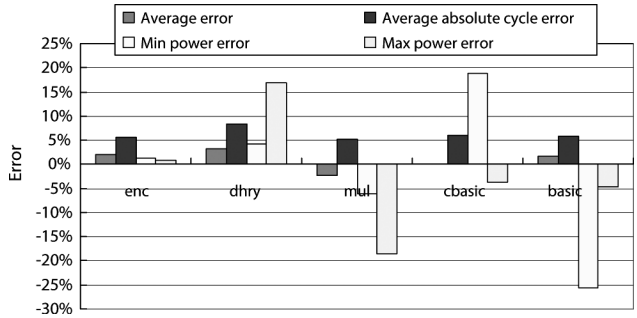


Fig. 17. Error comparison for various testbenches.

only level that relies on regression analysis, which is highly dependent on the tuning benchmark. Consequently, this level is expected to have the highest deviation in accuracy when tested with other benchmarks. From the figure it can be seen that not only is the average error for the testbenches fairly low, but the average absolute cycle error is lower than 6% for four out of the six benchmarks, and 8% at the most for "*dhry*". This is very close to the approximately 5% error obtained in Fig. 16, and shows how the power models created by our methodology are portable across multiple applications. The figure also shows interesting results for the corner cases of the minimum and maximum power estimation error at the system-level compared to gate-level power data. The worst error of the max power is about -18% for the "*mul*" testbench and the worst error of the min power is -25.5% for the "*basic*" test case. In general however, the corner case errors are much lower. The results from Fig. 17 show that our generated power model can provide reasonably acceptable power data for corner cases at the system level.

Fig. 18 shows a comparison between system level and gate level normalized power for the "*mul*" testbench executing on OpenRISC, across different ESL flow levels. The power data is normalized to the power of the idle status (NOP). Fig. 18 shows how the coarse grained *Level 1_b* instruction set model at the architectural/ISS level is unable to track the power variation very accurately due to the absence of a pipeline at that level. When the pipeline is captured, as in the *Level 2_b* case, then accuracy improves slightly. However, it requires a more detailed *Level 3_b* model which additionally captures the structural units in a cycle accurate manner, to accurately track the peaks of the gate level power waveform. The power estimated at this level can allow designers to accurately estimate peak power of the processor at simulation speeds that are 100–1000$\times$ faster than gate level power simulation. Such a model is extremely useful for determining the thermal and electrical limits of the design and can guide the selection of the appropriate packaging to prevent hotspots and thermal runaway.

B. MIPS Processor Power Modeling

To demonstrate the generality of our methodology, we applied it to a second processor case study using the MIPS ISA, a popular embedded platform used for many mobile and handheld applications. The MIPS platform has undergone many revisions in the ISA as well as in its bit-width. There are also several freely downloadable RISC processor models that use the MIPS instruction set. We chose Yet Another CPU CPU (YACC) from Opencores [25] for the second case study. The YACC MIPS has five pipeline stages as shown in Fig. 19. It has a 32-entry, 32-bit register file (RF) with two sub-banks storing the same register values to provide dual read ports. YACC has three execution units: arithmetic logic unit (ALU), Mul/Div, and Shifter in the execute pipeline stage. The pipeline is not interlocked for multiplication and division. Instead, the compiler is responsible for correct handling of these operations by inserting the necessary NOPs. This processor also has a UART unit for serial communication and a decoder unit to generate necessary the control signals. The processor has been synthesized on the Altera Stratix II FPGA yielding 110 DMIPS with an allowable clock frequency of 165 MHz. Several code applications written in C—including calculating 800 digits of Pi, a Reed Solomon error correction, and an interactive calculator—have been executed and tested on FPGA implementations (a Xilinx Spartan3 and Altera Cyclone) [25].

1) Experimental Results: Our goal was to create a power model for the MIPS processor using our proposed power model generation methodology that provides good accuracy ($<5\%$ absolute cycle error) while providing fast estimation at the system-level. Fig. 20 shows the distribution of power consumed in various units in the MIPS processor, determined using gate level power estimation. This breakdown is obtained by averaging the power results for five different testbenches used also in Fig. 22. As shown in Fig. 22, because each unit has a different power contribution, we need to have a better model for those units which have a larger influence, and can use a coarser grained model for other units which have less influence on overall power. The register file (RF) which consumes the largest power in the processor needs to be decomposed further into two separate banks for more detailed power modeling. Our power model for the MIPS has separate power LUTs for these decomposed units to account for situations where one bank is accessed at a time and consumes different amounts of power. The overall power model chosen for generation was a *Level 3* model, as defined in Section III-C. It has decomposed instructions, pipeline stages and structures sets and sub-decomposed detailed structural information for the separate banks of the RF. The waveform of the *Level 3* model is compared with the gate level simulation as shown in Fig. 21. The power is normalized to the power value of idle status. Fig. 21 shows how our system-level power model can give us power data that is close to data obtained from gate level simulation (with about 100$\times$ speedup).

The error for the *Level 3* model compared to the gate level power data for five test benchmarks is shown in Fig. 22. We note that the average and average absolute cycle errors, as well as the Min. power and Max. power errors are all quite low. The average error is less than 1% for four out of the five testbenches and the average of absolute cycle error is below 4% for all testbenches except "*pi*" which shows about 1.5% average error and about 5% average absolute cycle error, respectively. The Min. power error is below 10% for all cases and the Max. power error is below 6% for all the testbenches except "*pi*" which shows an 11.6% error. The minimum power data of the power model tends to be overestimated for all test cases because the processor consumes much less power during the warm up time, which

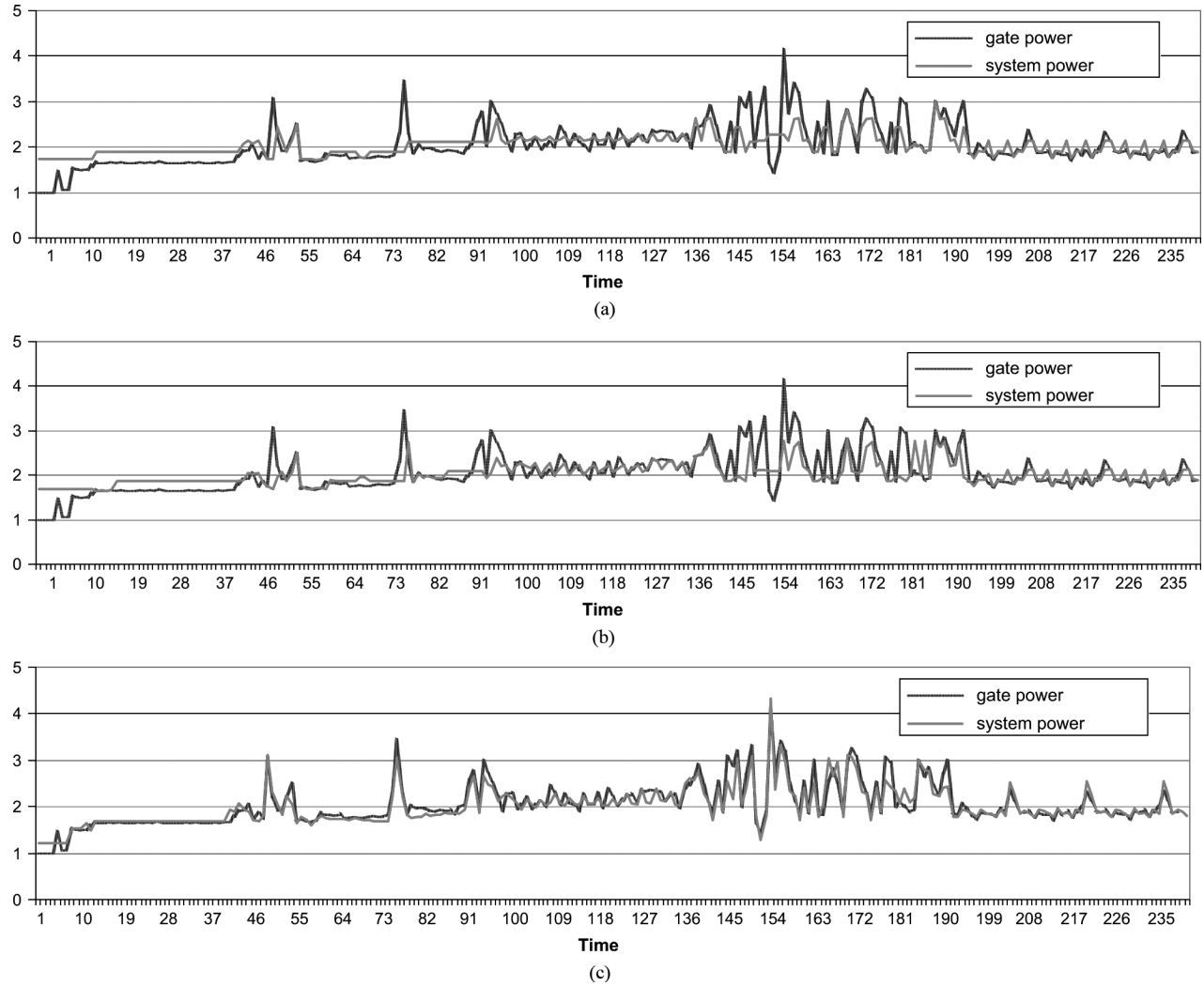


Fig. 18. Relative power waveform comparison for the "mul" testbench on OpenRISC (Unit for Time: 20 ns:) (a) Level 1_b; (b) Level 2_b; (c) Level 3_b.

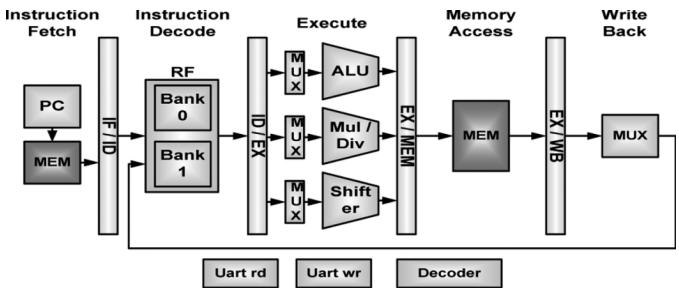


Fig. 19. MIPS architecture.

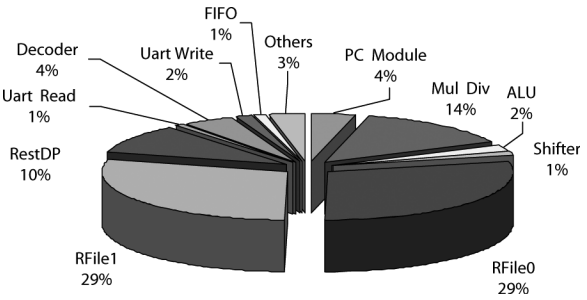


Fig. 20. Power consumption in various components of MIPS processor.

is not properly reflected in the power model. Since the power consumption behavior of this MIPS processor shows less variation compared to the OpenRISC processor case study, the developed power model after regression compensation shows relatively better accuracy for average error and average absolute error. However, it shows less accuracy for extreme corner cases (overestimating for min. power and underestimating for max. power). However, these results show that our power model can provide fairly good cycle accurate power data across various applications at the system-level, regardless of the specific ISA selected.

V. Modeling for ISA Customization

Today, there exist a number of customizable processor cores that can be embedded in a system-on-chip (SoC). These processors allow the SoC designer to add application specific instructions to perform certain applications more efficiently (or using

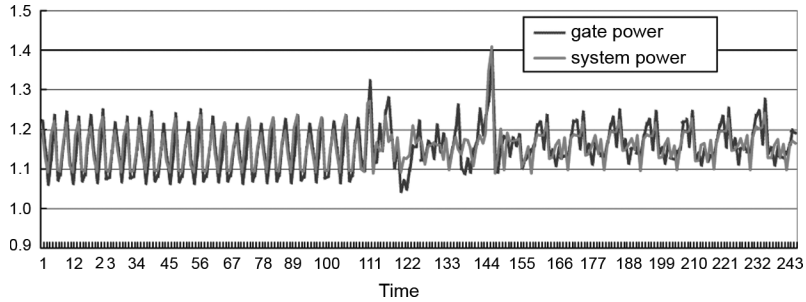


Fig. 21. Relative power waveform comparison of *Level 3* model for the "dhrystone" testbench on MIPS (unit for time: 20 ns).

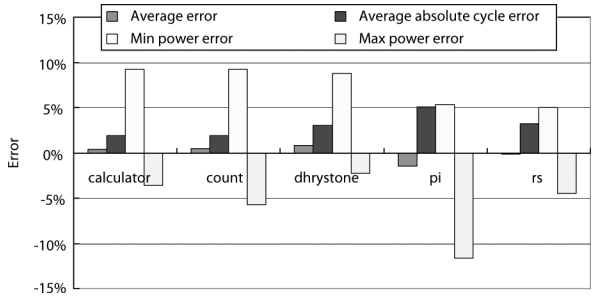


Fig. 22. Error comparison for MIPS power model.

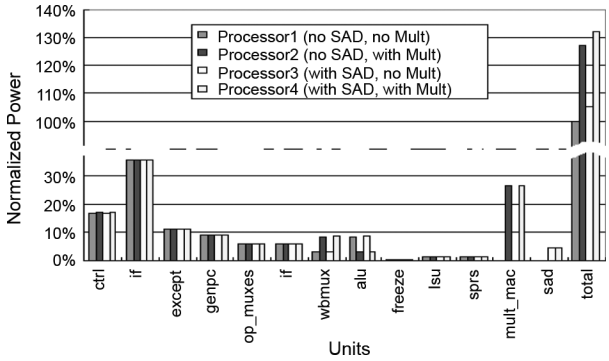


Fig. 23. Power contribution of units after adding SAD and/or MULT_MAC instruction set.

less energy). To generate power estimation models for such customizable processors, it becomes necessary to: 1) incrementally modify the power model to reflect the impact of the new instruction(s) and 2) estimate the change in power/energy at the software level using the new instruction(s). We show how this can be done with our methodology using two examples. The first example uses a multiply-accumulate (MAC) instruction that is already implemented in the original OpenRISC processor and the second example uses a newly added sum of absolute differences (SAD) instruction.

MAC is a commonly used operation in digital signal processing which computes the product of two numbers and adds that result to an accumulator $(\mathrm{A} \leftarrow \mathrm{A}+\mathrm{B}\times\mathrm{C})$. Many processors now contain a dedicated MAC unit consisting of a multiplier, an adder and an accumulator register which stores the output.

The sum of absolute differences (SAD) [also referred to as sum of absolute error (SAE)] is the most widely used metric for block-matching in motion estimation for video compression due to its computational simplicity [30]. It computes the absolute value of the difference between each pixel in the reference block and the corresponding pixel in the block being used for comparison. Then, these differences are summed to yield a metric for the comparison of block similarity. If the motion compensation block size is $N \times N$ samples, the SAD can be computed with the equation

$$\mathrm{SAD} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |C_{ij} - R_{ij}| \tag{8}$$

where C_{ij} is the sample of the current area to be compared and R_{ij} is the original reference area samples. Similar to the MAC unit, the implemented SAD unit has an absolute difference computing unit and an accumulator register which stores the temporary result when it is clocked. The output of the absolute difference computing unit is added to the register at each clock cycle without storing the output to the general registers. For this purpose, the following two instructions are added to the instruction set: 1) l.sad rA, rB and 2) l.sadrc rD. The l.sad instruction computes the absolute difference of the contents of general-purpose register rA and rB, and the result is added to the temporary special-purpose register. Once all instructions in the SAD pipeline are completed, the l.sadrc instruction places the contents of the special-purpose register in the SAD unit into the general-purpose register rD and the SAD accumulator is cleared.

A. Impact on Power Modeling

Fig. 23 compares four possible customizations of the OpenRISC processor, and their impact on the power contribution of various functional units. The first customization is a processor having neither MULT_MAC unit (which is for multiplication, and MAC instruction) nor SAD unit (processor 1), the second has MULT_MAC but no SAD (processor 2), the third implementation has SAD but no MULT_MAC (processor 3) and the last implementation case has both MULT_MAC and SAD units (processor 4).

Fig. 23 shows the power contribution for various functional units for these four cases, normalized to the total power of processor 1, which is 100%. Fig. 23 shows that the total power increase of processor 2 (27.13%), processor 3 (5.20%), and processor 4 (32.14%) comes from their additional units, MULT_MAC and SAD. The MULT_MAC causes about 27% and SAD causes about 5% increase of power respectively. We observe from the figure that the influence of adding new instructions and their corresponding circuitry does not influence power for other units if the functional units for the new

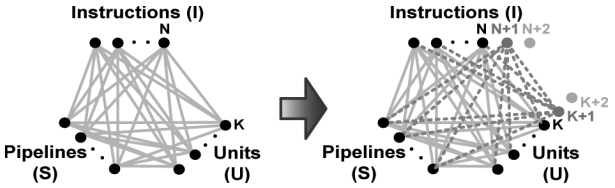


Fig. 24. Changes of tripartite hyper-graph after adding extra instructions.

TABLE I
IMPACT OF ADDING SAD INSTRUCTION

	Instructions	Time (Cycles)	Power (W)	Energy (J)
1. no SAD	2304	16096	6.70E-04	0.108u
2. with SAD	224	2400	6.96E-04	0.017u
difference	-90.28%	-85.09%	3.94%	**-84.50%**

instructions are well separated from the rest of the units, as is the case here. This separation of functional units is also quite helpful for saving effort in building the processor power model as shown in Fig. 24. If we assume that the number of units in processor 1 is K, for processor 2 and processor 3 is $K+1$, and for processor 4 is $K+2$, then since we know that the influence of new instruction on other units is minimal, we have to extend our power modeling methodology only for these additional one or two extra unit(s), while the other unit's power data is reusable without significantly influencing overall power accuracy. Similarly, if we assume that the original number of instruction is N, we can reuse the previous power data for the original instructions and create power models only for the newly added instructions $N+1$, (and $N+2$, if necessary) as shown in Fig. 24.

B. Impact of ISA Customization

While adding instructions is expected to increase the hardware and therefore power consumption, it has significant impact on the code size and therefore the overall energy. In order to quantify this impact, we implemented two versions of a simple block-matching algorithm [30] on the OpenRISC processor. The first runs on the customized processor 2 (no SAD implementation) and the second runs on the customized processor 4 (with SAD implementation). The impact of adding the SAD instruction is shown in Table I. As expected, the SAD instruction in processor 4 can significantly reduce the total number of compiled instructions (90.28% reduction of instructions compared to the processor 2) and also saves overall execution time which is now 85.09% less running time (clock cycles) than processor 2. We observed about 4% of increase in power consumption because of the added circuitry of the SAD unit. However, due to the reduced execution time, the energy consumed by processor 4 is now 84.5% less than that of consumed by processor 2 for the same computation.

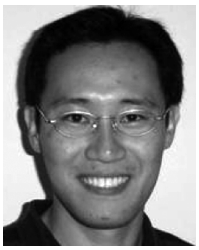
VI. CONCLUSION

In this paper we presented a multigranularity processor power modeling methodology for generating various power models that can be used at different stages in an ESL design flow. We introduced a tripartite hyper-graph representation of the processor instruction set, pipeline, and functional units. Iteratively decomposing the hyper nodes in this graph allows our methodology to create increasingly more detailed power models with better accuracy. Ultimately, our methodology can allow designers to generate processor power models for any stage of an ESL design flow, with support for model customization that allows tradeoffs between simulation speed and estimation accuracy. The generated power models enable designers to explore the power design space and determine the effect of using various power optimization techniques early in the design flow. Our ongoing work is focusing on integrating floorplanning and routing information at the system level, for even more accurate early processor power estimation.

REFERENCES

[1] Y.-H Park, S. Pasricha, F. Kurdahi, and N. Dutt, "Methodology for multi-granularity embedded processor power model generation for an ESL design flow," in *Proc. CODES + ISSS*, 2008, pp. 255–260.
[2] I. Lee, H. Kim, P. Yang, S. Yoo, E. Chung, K. Choi, J. Kong, and S. Eo, "PowerVip: SoC power estimation framework at transaction level," in *Proc. ASP-DAC*, Jan. 2006, pp. 551–558.
[3] W. Nebel, "System-level power optimization," in *Proc. DSD*, Sep. 2004, pp. 27–34.
[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," *Proc. ISCA*, pp. 83–94, 2000.
[5] 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.
[6] Hewlett-Packard Laboratories, Palo Alto, CA, "Cacti5.3," 2008. [Online]. Available: http://quid.hpl.hp.com:9081/cacti/
[7] N. Banerjee, P. Vellanki, and K. S. Chatha, "A power and performance model for network-on-chip architectures," *Proc. DATE*, vol. 2, pp. 1250–1255, Feb. 2004.
[8] Y.-H. Park, S. Pasricha, F. Kurdahi, and N. Dutt, "System-level power estimation methodology with H.264 decoder prediction IP case study," *Proc. ICCD*, pp. 601–608, Oct. 2007.
[9] V. Tiwari, S. Malik, A. Wolfe, and M. T.-C. Lee, "Instruction level power analysis and optimization of software," *J. VLSI Signal Process. Syst.*, vol. 13, no. 2–3, pp. 223–238, 1996.
[10] H. Mehta, R. M. Owens, M. J. Irwin, R. Chen, and D. Ghosh, "Techniques for low energy software," *Proc. ISLPED*, pp. 72–75, Aug. 1997.
[11] C. H. Gebotys and R. J. Gebotysu, "An empirical comparison of algorithmic, instruction, and architectural power prediction model for high-performance embedded DSP processors," in *Proc. ISLPED*, Aug. 1998, pp. 121–123.
[12] D. Sarta, D. Trifone, and G. Ascia, "A data dependent approach to instruction level power estimation," in *Proc. IEEE AVMW Low-Power Design*, Mar. 1999, pp. 182–190.
[13] C. Chakrabarti and D. Gaitonde, "Instruction level power model of microcontrollers," in *Proc. ISCAS*, Jun. 1999, vol. 1, pp. 76–79.
[14] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "Instruction-level power estimation for embedded VLIW cores," in *Proc. CODES*, Mar. 2000, pp. 34–38.
[15] N. Kavvadias, P. Neofotistos, S. Nikolaidis, K. Kosmatopoulos, and T. Laopoulos, "Measurements analysis of the software-related power consumption in microprocessors," in *Proc. IMTC*, May 2003, vol. 2, pp. 981–986.
[16] A. Varma, E. Debes, I. Kozintsev, and B. Jacob, "Instruction-level power dissipation in the Intel XScale embedded microprocessor," in *Proc. SPIE's Annu. Symp. Electron. Imag. Sci. Technol.*, Jan. 2005, vol. 5683, pp. 1–8.
[17] M. Schneider, H. Blume, and T. G. Noll, "Power estimation on functional level for programmable processors," *Adv. Radio Sci.*, pp. 215–219, 2005.
[18] D. Burger and T. M. Austin, "The SimpleScalar tool set, version 2.0," *ACM SIGARCH Comput. Arch. News*, vol. 25, no. 3, pp. 13–25, Jun. 1997.
[19] OpenCores, "OpenRISC 1000 Family," 2008. [Online]. Available: http://www.opencores.org/projects/or1k/
[20] J. J. Faraway, *Linear Models with R*. Boca Raton, FL: CRC, 2004.

[21] Synopsys, Mountain View, CA, "Synopsys Design Compiler, PrimeTime PX, Power Compiler," 2007. [Online]. Available: http://www.synopsys.com
[22] Cadence, San Jose, CA, "Cadence NC-Verilog ," 2008. [Online]. Available: http://www.cadence.com
[23] Open SystemC Initiative (OSCI), "SystemC initiative," 1999. [Online]. Available: http://www.systemc.org
[24] S. Pasricha, Y.-H Park, F. Kurdahi, and N. Dutt, "System-level power-performance trade-offs in bus matrix communication architecture synthesis," in *Proc. CODES + ISSS*, Oct. 2006, pp. 300–305.
[25] OpenCores, "YACC," 2005. [Online]. Available: http://www.opencores.org/projects.cgi/web/yacc/
[26] T. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and minimization techniques for embedded DSP software," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 5, no. 1, pp. 123–135, Mar. 1997.
[27] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, "Reducing the complexity of instruction-level power models for VLIW processors," *Des. Autom. for Embedded Syst.*, vol. 110, no. 1, pp. 49–67, Mar. 2005.
[28] P. Yu and P. Schaumont, "Secure FPGA circuits using controlled placement and routing," *Proc. CODES + ISSS*, pp. 45–50, 2007.
[29] Y. Bonhomme, P. Girard, C. Landrault, and S. Pravossoudovitch, "Test power: A big issue in large SoC designs," *Proc. DELTA*, pp. 447–449, Jan. 2002.
[30] I. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. New York: Wiley, 2003.
[31] L. Benini and G. D. Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan. 2002.
[32] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-system chip multiprocessor power evaluations using FPGA-based emulation," in *Proc. ISLPED*, Aug. 2008, pp. 335–340.



Young-Hwan Park (M'07) was born in Seoul, Korea. He received the Ph.D. degree in electrical engineering and computer science from the University of California, Irvine, in 2009.

He is currently an R&D staff member with the Samsung Advanced Institute of Technology, Gyeonggi-do, Korea, where he is engaged in research on software defined radio and reconfigurable processor. He has six years of work experience at Samsung as a Hardware Design Engineer and intern experience in NVIDIA, where he was involved in designing a new graphic processing unit in 2007, and Qualcomm, where he was engaged in system level design exploration and modeling of next generation SoC in 2008. He is the author or coauthor of over 10 research papers in conferences and journals. His current research interests include the areas of embedded processor/DSP design, computer architecture, communication and multimedia SoC implementation, and system-level modeling methodology.

Dr. Park was a corecipient of the 2006 International Symposium on Quality Electronic Design (ISQED) Best Paper Award and the Semiconductor Research Corporation (SRC) fellowship from 2005 to 2008.

Sudeep Pasricha (M'02) received the B.E. degree in electronics and communication engineering from Delhi Institute of Technology, Delhi, India, in 2000, and the M.S. and Ph.D. degrees in computer science from the University of California, Irvine, in 2005 and 2008, respectively.

He is currently an Assistant Professor with Colorado State University, Fort Collins, with a joint appointment in the Department of Electrical and Computer Engineering, and Department of Computer Science. His research interests include the areas of multiprocessor embedded system design, networks-on-chip and emerging nanophotonic, carbon nanotube, and 3-D interconnect technologies, system-level modeling languages and design methodologies, and computer architecture.

Dr. Pasricha is currently serving in the program committee of several premier conferences including CODES + ISSS, NOCS, ISQED, GLSVLSI, and VLSI Design. He was a recipient of the 2006 ASPDAC Best Paper Award, a Best Paper Award nomination at DAC 2005, and several fellowships and awards for excellence in research.

Fadi J. Kurdahi (M'87–SM'03–F'05) was born in Beirut, Lebanon. He received the B.Eng. degree from the American University of Beirut, Beirut, Lebanon, in 1981, and the M.S.E.E. and Ph.D. degrees from the University of Southern California, Los Angeles, in 1982 and 1987, respectively.

Since 1987, he has been a Faculty Member with the Electrical Engineering and Computer Science and Integrated Circuit Systems Departments at the University of California, Irvine, where he is engaged in research on VLSICAD and reconfigurable computing with applications in communication and multimedia systems.

Dr. Kurdahi served as the Program Chair and General Chair of the 1999 and 2000 International Symposium of System Synthesis. He also served on executive committees, steering committees, and technical program committees of numerous IEEE conferences. He was an Associate Editor for the IEEE Transactions on Circuits and Systems II 1993–1995, and Topic Area Editor for reconfigurable computing for IEEE Design and Test (2000–present). He was the corecipient of the 2001 IEEE Transactions on Very Large Scale Integrated (VLSI) Systems Best Paper Award and the 2006 ISQED Best Paper Award, as well as three Distinguished Paper Awards at DAC, ASPDAC, and EuroDAC.

Nikil D. Dutt (M'84–SM'96–F'08) received the Ph.D. degree in computer science from the University of Illinois at Urbana-Champaign, Urbana-Champaign, in 1989.

He is currently a Chancellor's Professor with the University of California, Irvine, with academic appointments in the Computer Science and Electrical Engineering and Computer Science Departments. His research interests include embedded systems, electronic design automation, computer architecture, optimizing compilers, system specification techniques, distributed embedded systems, formal methods, and brain-inspired architectures, and computing.

Prof. Dutt was a recipient of Best Paper Awards at CHDL89, CHDL91, VLSIDesign2003, CODES + ISSS 2003, CNCC 2006, ASPDAC-2006, and IJCNN 2009. He currently serves as an Associate Editor of *ACM Transactions on Embedded Computer Systems*, and IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. He served as Editor-in-Chief of *ACM Transactions on Design Automation of Electronic Systems* between 2004–2008. He was an ACM SIGDA Distinguished Lecturer during 2001–2002 and an IEEE Computer Society Distinguished Visitor for 2003–2005. He has served on the steering, organizing, and program committees of several premier CAD and Embedded System Design conferences and workshops, including DAC, DATE, ESWEEK, ICCAD, ISLPED, RTAS, and RTSS. He serves or has served on the advisory boards of ACM SIGBED and ACM SIGDA, the ACM Publications Board, and previously served as Vice-Chair of ACM SIGDA and of IFIP WG 10.5. He is an ACM Distinguished Scientist and an IFIP Silver Core Awardee.