Methodology for Multi-Granularity Embedded Processor Power Model Generation for an ESL Design Flow

Young-Hwan Park†, Sudeep Pasricha*, Fadi J. Kurdahi†, Nikil Dutt†

†University of California, Irvine, CA
{younghwp, kurdahi, dutt}@uci.edu

*Colorado State University, Fort Collins, CO
sudeep@engr.colostate.edu

ABSTRACT

With power becoming a major constraint for multi-processor 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 an OpenRISC processor demonstrate the usefulness of having multiple power models. The generated models range from very high-level two-state and architectural/ISS models that can be used in transaction level models (TLM), 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.

Categories and Subject Descriptors: J.6 [**Computer-aided Design**]; B.7.2 [**Integrated circuits**]: Design; C.5.4 [**VLSI Systems**]
General Terms: Design, Experimentation, Performance
Keywords: Embedded Processor, Power Modeling, System-on-Chip, ESL

1. INTRODUCTION

Reducing power dissipation is a critical design goal for electrical devices from hand-held systems with limited battery capacity to large computer workstations that dissipate huge amounts of power and need 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 [1]-[2]. 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 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.

Several system level power estimation approaches have been proposed in recent years focusing on the various components of CMP designs, such as processors [3]-[4], memories [5], interconnection fabrics [6], and custom ASIC blocks [7]. 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.

CODES+ISSS'08, October 19-24, 2008, Atlanta, Georgia, USA.

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 [8] 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 [3] and SimplePower [4] 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.

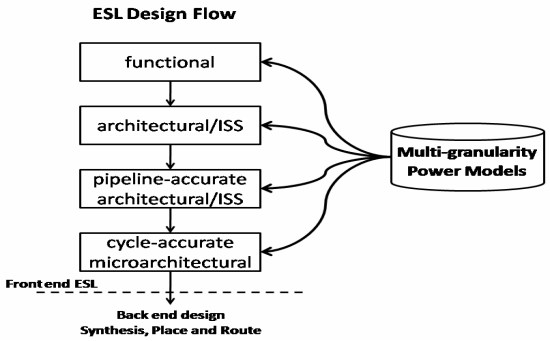


Figure 1: ESL design flow for embedded processors

In this paper, we propose a comprehensive multi-granularity power model generation methodology that spans the entire ESL design flow (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, with the aid of 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 a typical ESL design flow. We demonstrate the feasibility of our approach on an OpenRISC [18] processor case study, and present results to show how the multi-granularity power models generated for the processor provide designers with the flexibility to trade-off estimation accuracy and simulation effort during system-level exploration.

2. RELATED WORK

Processor power estimation has been the focus of several research efforts over the past few 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 [1]. A popular power estimation technique for processors is instruction level power estimation, which was first proposed by Tiwari et al. [8]. The technique is based on the simple observation that each instruction can be assigned its own power cost. Several subsequent research efforts [9]-[14] have applied instruction level power estimation to obtain high level power estimates for various processor variants such as a DSPs [10], VLIW processors [13] and the Intel XScale [15]. However, while these 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 [3][4][16] 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 [3] and SimplePower [4] use this approach, and are relatively well known because they can cooperate with the widely used processor simulation platform SimpleScalar [17]. Even though these tools are popular, they have their limitations. Wattch concentrates its effort 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.

All of these processor power estimation techniques have trade-offs between designer effort, accuracy and simulation speed. To the best of our knowledge, there has not been any approach that can comprehensively generate several 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 multi-granularity models by simply varying certain parameters to trade-off 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 large number of processors, and facilitates the realistic optimization of such systems. Finally, we note that while our approach concentrates on the processor cores only, it is complementary to existing memory models such as CACTI[5] which provide estimates of caches and other memories peripheral to the processor core that are comparable in terms of accuracy and flexibility to our models.

3. POWER MODELING METHODOLOGY

In this section we present details of our power estimation methodology.

3.1 3D 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 3D lookup table (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 all of the functional units in the processor. For improved accuracy, we create a set of three 3D 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. Fig. 3 shows the normalized power for several instructions from the OpenRISC ISA [18], for varying HD values of operand data. It can be seen that power dissipation between the minimum and maximum HD cases varies by as much as 3×, justifying the need for three 3D LUTs. Linear interpolation is used to estimate the power entries in the LUTs for arbitrary data values using their HD. Section 3.5 describes how we populate the 3D LUTs in more detail, and how inter-instruction and other unpredictable power dissipation factors are captured using regression compensation.

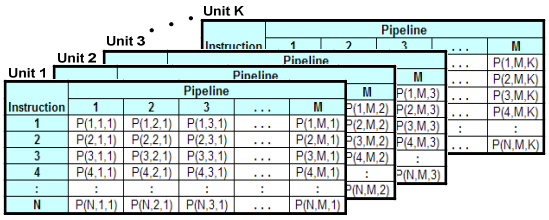


Figure 2. 3D Power contribution LUT

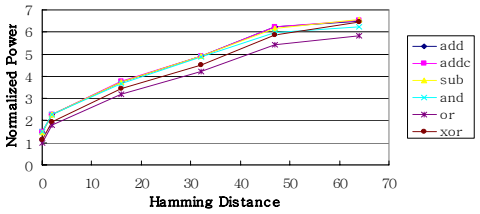


Figure 3. Relation between Hamming Distance and power of several instructions at EX stage in ALU unit of the OpenRISC processor

3.2 Processor Tripartite Hyper-Graph

For the purpose of creating multi-granularity processor power models, we can conceptually represent a processor as a tripartite hyper-graph $H(P) = <V,E>$, 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. The ternary edges (or hyperedges) E is a set of triples (i,s,u), $i \in I$, $s \in S$, $u \in U$ that represent the ternary association between an instruction and its power dissipation across the pipeline stages it traverses and functional units it activates. The weights of those hyperedges are populated from the 3D 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 $1 \le |I'| \le N$, $1 \le |S'| \le M$ and $1 \le |U'| \le K$ clusters for each of the three sets of vertices. The number of hyperdeges is also reduced as one hyperedge 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 2-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)-4(d). These models will more accurately represent the power dissipation of multiple instructions *I*, as they traverse the pipeline stages *S,* and activate *U* functional units.

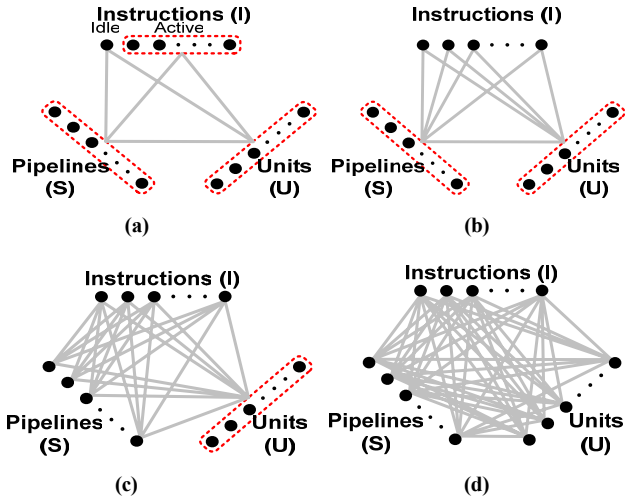


Figure 4. Tripartite hyper-graph H(P), (a) simplest 2-state power model, (b) power model with set I decomposed, (c) power model with sets I, S decomposed, (d) power model with sets I, S, U decomposed

3.3 Processor Power Models for an ESL Flow

Having introduced the concept of a tripartite hyper-graph that allows multi-granularity 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 four power models that map to the appropriate four major ESL stages:

Level 0 (functional): A 2-state (active, idle) coarse grained power model is used for the functional stage, and is shown in Fig. 4(a).

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 ($|S'| = 1$, $|U'| = 1$).

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.

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

3.4 Power Model Customization

The multi-granularity 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 trade-off 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'*. Due to lack of space, we only present a brief overview of the approaches in the following subsections. More information on these algorithms can be found in our detailed technical report [19].

3.4.1 Instruction Set Clustering

One possible power model customization is to reduce the number of instructions considered for power estimation, by clustering similar power dissipating instruction 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 3D 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 *I2* is smaller than the threshold value ($|P_{max}(I1)-P_{max}(I2)| < T_d$ and $|P_{min}(I1)-P_{min}(I2)| < T_d$), we can regard these instructions as similar enough to be clustered together. 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.

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 the example in Fig. 5, $T_r > P_{max}(G3)-P_{min}(G3)$, and therefore for *G3*, min and max values are discarded, and we just keep an average value. For group *G2*, $T_r < P_{max}(G2)-P_{min}(G2)$, 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.

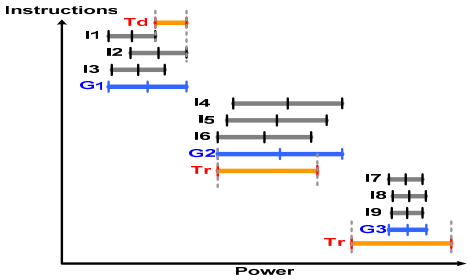


Figure 5. Instructions and clustering groups

3.4.2 Further Decomposition and Regression Analysis

It is possible that greater accuracy is required than the accuracy supported by the models described in Section 3.3. In such a case, we can increase 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.

If such decomposition is still unable to provide sufficient accuracy, we can then perform a regression compensation step. In this step, we account for hard to determine factors that contribute to power dissipation, including any complex inter-instruction influences due to multiple instructions traveling at the same time through the pipeline. 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, based on testbench simulation and subsequent curve fitting [20] can be expressed as:

$$P_{Cycle} = \alpha_0 + \sum_{s=1}^{M} \alpha_s \cdot P(i,s,u)$$

where M is the number of pipeline stages in the processor, α_0 is regression coefficient representing power of the factors that are independent of the model variables, α_s is the regression coefficient for $P(i,s,u)$, which is the power value from LUT for instruction i, pipeline stage s and unit u. Our experimental results show that such a compensation step can improve estimation accuracy noticeably.

3.5 Power Model Generation Methodology

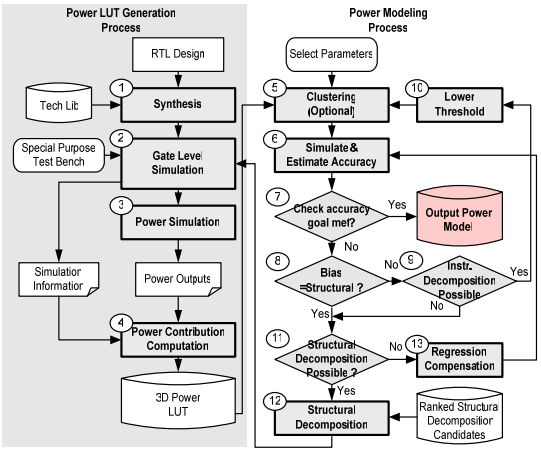


Figure 6. Power model generation methodology

The overall methodology to build the multi-granularity power models for an ESL design flow is shown in Fig. 6. The methodology consists of two major flows: the 3D 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. This testbench consists of all the instructions in the ISA separated by an appropriate number of NOPs, to isolate the power dissipation for each instruction type. 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 simple Perl scripts. The generated information in Steps 2-3 is provided to Step 4, which generates the 3D LUTs.

The generated 3D 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 3.3), the appropriate values to create the tripartite hyper-graph are specified. The target accuracy goal for the power model is also specified. Additionally, if a trade-off 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 (Section 3.4.1). 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 (Section 3.4.2) and repeat the flow from Step 6 onwards, till the output power model with appropriate accuracy is generated.

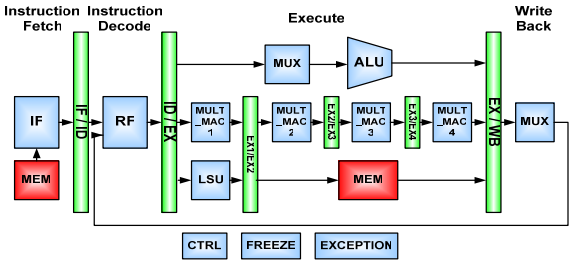


Figure 7. CPU/DSP Core architecture of OR1200

4. OpenRISC Processor Power Modeling

To evaluate the effectiveness of our multi-granularity power model generation methodology, we use the OR1200 freely downloadable open source RISC processor (part of the OpenRISC 1000 family [18]) as a case study. OR1200 is a 32-bit scalar RISC processor with a Harvard architecture. Fig. 7 shows a basic block diagram of the processor, which has a 4 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 (4 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 300MHz and 300 DSP MAC 32x32 operations that is at least 20% better than other competitors in this class (32-bit RISC processors) such as ARM10 and Tensilica RISC processors [18].

4.1 OpenRISC Power Models

Fig. 8 shows the multi-granularity 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 3.3. Three additional levels are also created, customized for the

OpenRISC, to further trade-off 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.

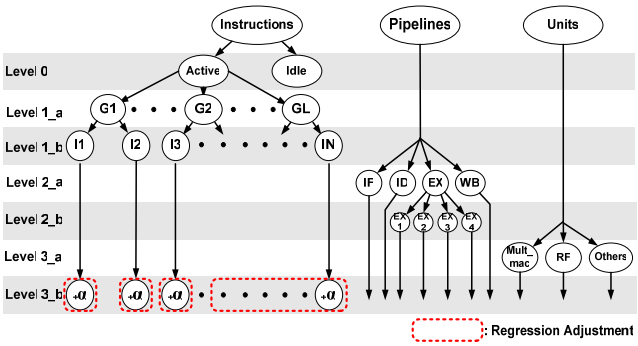


Figure 8. Hierarchical power model for OpenRISC processor

5. EXPERIMENTAL RESULTS

In this section we present results of applying our methodology on the OpenRISC processor case study. The power models in Fig. 8 were generated using our proposed methodology in Fig. 6, 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 2.1 [23] for the four levels of modeling abstraction in an ESL design flow (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 multi-granularity ESL power models, when compared to gate level power can be calculated as:

$$E^i_{AC} = \frac{\left|P^i_S - P^i_G\right|}{P^i_G} \times 100\%$$

where E^i_{AC} is absolute cycle error at cycle i, P^i_S is cycle power obtained from system level simulation with the generated power model, and P^i_G 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:

$$E_{AAC} = \frac{\sum_{i=1}^{N} E^i_{AC}}{N}$$

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

Fig. 9 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 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. 9 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 don't 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 trade-off 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.

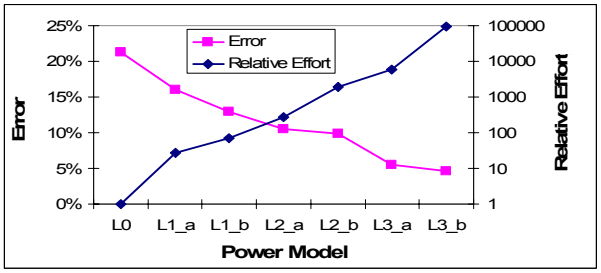


Figure 9. Average absolute cycle error for power models

The results in Fig. 9 are presented for the tuning benchmark (Section 3.5) 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. 10 shows the average error and average absolute cycle error for the Levels 3_b model, compared to gate level estimates, for several benchmarks. We chose Level 3_b for the comparison because it is the 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. 9, and shows how the power models created by our methodology are portable across multiple applications.

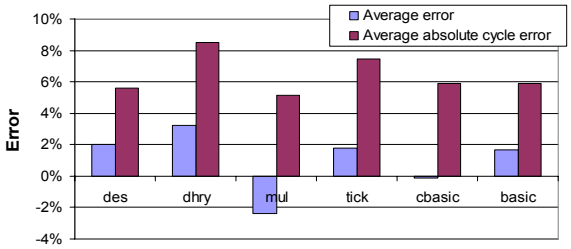
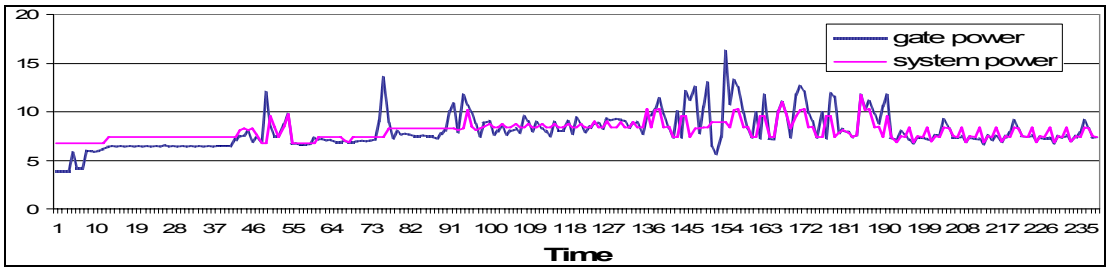
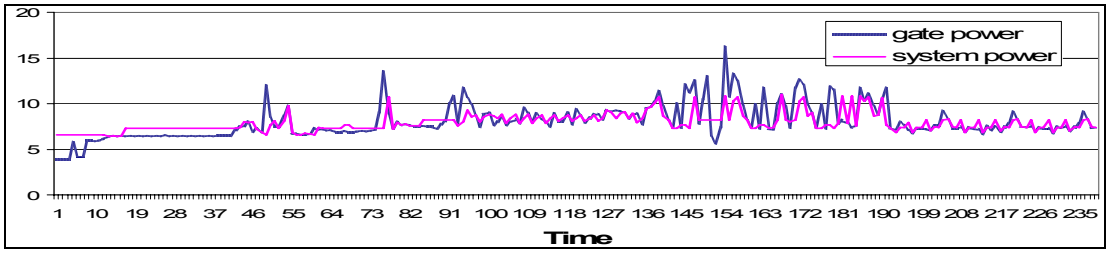


Figure 10. Error comparison for various testbenches

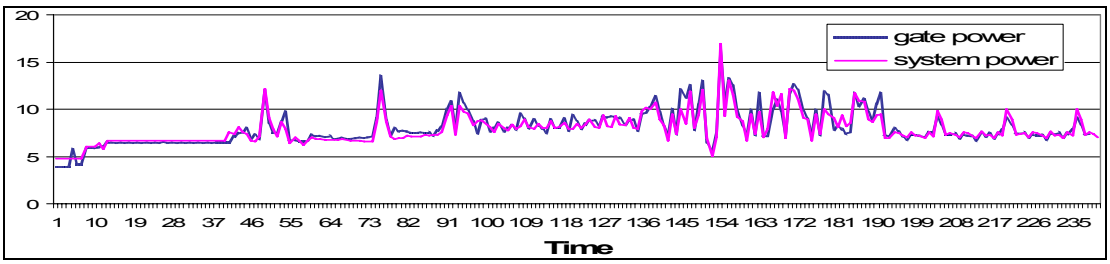
Fig. 11 shows a comparison between system level and gate level normalized power for the 'mul' testbench executing on OpenRISC, across different ESL flow levels. The figure shows how the coarse grained *level 1_b* instruction set model at the architectural/ISS level is unable to track the power variation 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× 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.



(a) level 1_b



(b) level 2_b



(c) level 3_b

Figure 11. Relative power waveform comparison for the 'mul' testbench on OpenRISC (Unit for Time: 20ns)

6. CONCLUSION

In this paper we presented a multi-granularity 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 trade-offs 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 optimizations, early in the design flow. Our ongoing work is focusing on applying this methodology to other types of processors, and integrating floorplanning and routing information at the system level, for even more accurate early processor power estimation.

7. ACKNOWLEDGMENTS

This research was partially supported by grants from SRC (2005-HJ-1330 and 1617.001) and NSF (CCF-0702797).

REFERENCES

[1] I. Lee, et al, "PowerViP: Soc power estimation framework at transaction level", *Proc. ASP-DAC* 2006.
[2] W. Nebel, "System-Level Power Optimization", *Proc. DSD* 2004.
[3] D. Brooks, et al, "Wattch: a framework for architectural-level power analysis and optimizations", *Proc. ISCA*, pp. 83-94, 2000.
[4] W. Ye, et al, "The design and use of SimplePower: a cycle-accurate energy estimation tool", *Proc. DAC* 2000.
[5] "Cacti4", http://quid.hpl.hp.com:9081/cacti/.
[6] N. Banerjee, et al, "A power and performance model for network-on-chip architectures", *Proc. DATE* 2004.
[7] Y. Park, et al, "System-level power estimation methodology with H.264 decoder prediction IP case study", *Proc. ICCD* 2007.
[8] V. Tiwari, et al, "Instruction Level Power Analysis and Optimization of Software", *Journal of VLSI Signal Processing*, pp.223-233, 1996.
[9] H. Mebta, et al, "Techniques for Low Energy Software", *International Symposium on Low Power Electronics and Design*, pp. 72-75, Aug 1997.
[10] C. Gebotys, et al, "An Empirical Comparison of Algorithmic, Instruction, and Architectural Power Prediction Model for High-Performance Embedded DSP Processors", *Proc. ISLPED* 1998.
[11] D. Sarta, et al, "A data dependent approach to instruction level power estimation", *Proc. IEEE AVMW Low-Power Design*, Mar. 1999.
[12] C. Chakrabarti et al, "Instruction level power model of microcontrollers", *Proc. IEEE ISCAS*, pp. 176-179, 1999.
[13] M. Sami, et al, "Instruction-level power estimation for embedded VLIW cores", *Proc. CODES* 2000.
[14] N. Kavvadias, et al, "Measurements analysis of the software-related power consumption in microprocessors", *Proc. IMTC*, pp. 981- 986, 2003.
[15] A. Varma, et al, "Instruction-level power dissipation in the Intel XScale embedded microprocessor." *Proc. SPIE's 17th Annual Symposium on Electronic Imaging Science & Technology*, Jan.2005.
[16] M. Schneider, et al, "Power estimation on functional level for programmable processors", *Advances in Radio Science*, pp 215-219, 2005.
[17] D. Burger , et al, "The simplescalar tool set, version 2.0", Technical report, Computer Sciences Dept., University of Wisconsin, June, 1997.
[18] OpenRISC 1000 Family, http://www.opencores.org/projects/or1k/.
[19] Y. Park, et al, "Methodology for Multi-Granularity Embedded Processor Power Model Generation", UCI Technical Report, 2008.
[20] J.J. Faraway, "Linear Models with R", *CRC Press,* 2004.
[21] Synopsys Design Compiler, PrimeTime PX, http://www.synopsys.com.
[22] Cadence NC-Verilog, http://www.cadence.com.
[23] SystemC initiative, http://www.systemc.org.
[24] S. Pasricha, et al, "System-level power-performance trade-offs in bus matrix communication architecture synthesis", *Proc. CODES+ISSS* 2006.