Dynamic Power and Performance Back-Annotation for Fast and Accurate Functional Hardware Simulation

Dongwook Lee, Lizy K. John, and Andreas Gerstlauer
Department of Electrical Computer Engineering
The University of Texas Austin
{dongwook.lee,ljohn,gerstl}@utexas.edu

***Abstract*—Virtual platform prototypes are widely used for early design space exploration at the system level. There is, however, a lack of accurate and fast power and performance models of hardware components at such high levels of abstraction. In this paper, we present an approach that extends fast functional hardware models with the ability to produce detailed, cycle-level timing and power estimates. Our approach is based on back-annotating behavioral hardware descriptions with a dynamic power and performance model that allows capturing cycle-accurate and data-dependent activity without a significant loss in simulation speed. By integrating with existing high-level synthesis (HLS) flows, back-annotation is fully automated for custom hardware synthesized by HLS. We further leverage state-of-the-art machine learning techniques to synthesize abstract power models, where we introduce a structural decomposition technique to reduce model complexities and increase estimation accuracy. We have applied our back-annotation approach to several industrial-strength design examples under various architecture configurations. Results show that our models predict average power consumption to within 1% and cycle-by-cycle power dissipation to within 10% of a commercial gate-level power estimation tool, all while running several orders of magnitude faster.**

I. Introduction

The continued rise in hardware and software complexities of embedded on-chip systems has necessitated the elevation of the design process to higher levels of abstraction. Virtual platform models capable of simulating whole systems are widely employed to provide rapid feedback for design space exploration. Instead of slow co-simulation with low-level RTL or cycle-accurate models of accelerators and other IPs, a purely functional modeling of hardware behavior is typically utilized. Furthermore, high-level synthesis (HLS) is used to translate such behavioral hardware models into final implementations. This provides the ability to perform early exploration with an automated path to implementation. However, the modeling gap between virtual platforms and physical hardware implementations severely limits their usefulness. To support efficient exploration, there is a need for hardware models that can provide quick yet accurate estimates of critical system metrics such as performance and power at a high level of abstraction.

Most previous work in high-level hardware modeling has relied on fast yet coarse-grain estimation using state-based models [1, 2, 3, 4]. Other approaches use accurate but slow activity estimation at a fine-grain micro-architecture or register-transfer level [5, 6, 7, 8, 9, 10]. More recently, solutions at the intermediate representation (IR) level have emerged [11, 12, 13]. However, they still rely on fine-grain simulation of the cycle-by-cycle behavior of individually scheduled IR operations in control/dataflow graph (CDFG) or finite state machine with data (FSMD) form to obtain accurate results.

Existing approaches all simulate hardware functionality at the same level of detail at which performance and power is modeled. This allows the functional simulation to drive an accurate, potentially data-dependent estimation model, but it also creates a fundamental tradeoff between speed and accuracy depending on the simulation granularity. By contrast, we propose a novel approach that is aimed at bridging the modeling gap by dynamically back-annotating a high-level functional model of hardware behavior with low-level power and performance estimates. Instead of detailed micro-architecture or FSMD/CDFG simulation, we statically synthesize a cycle-accurate and data-dependent switching activity model of a given gate-level implementation using machine learning approaches. Based on resource scheduling and binding information, traces of operand and result value transitions captured from a functional IR simulation are then used to drive the abstracted offline or online power-performance model. In contrast to prior work, this enables cycle-accurate, data-dependent estimation at the speed of a fast functional simulation.

The main contribution of this paper is a hardware modeling framework that realizes such a novel, fast yet accurate white-box back-annotation approach. Our framework integrates with existing, commercial HLS tools to provide a fully automated, HLS-driven back-annotation process. Generated hardware models are compatible with and can be seamlessly integrated into existing SystemC/TLM based virtual platforms or other system architecture simulators. Within our framework, we make the following specific contributions: (1) We develop a light-weight approach for extracting cycle-accurate signal transition information from a high-level functional simulation without the need for full architecture simulation; (2) We propose a novel concept for dynamic back-annotation of fast functional models with abstract, data- and cycle-dependent power models generated using machine learning approaches; (3) We introduce a novel approach for decomposing learning-based power models using scheduling and binding information to reduce model complexity while improving estimation accuracy; (4) To further reduce the overhead of extracting switching activity information, learning-based feature selection is applied to each decomposed power model.

The rest of the paper is organized as follows: following a discussion of related work, problem definition and an overview of our back-annotation flow, Sections II and III elaborate on each step of our methodology. Section IV shows experimental results of applying the flow to a set of industrial-strength design examples. Finally, Section V concludes the paper with a summary and an outlook on future work.

A. Related Work

To generate higher-level timing and energy models of custom hardware processors, library- or learning-based approaches can be utilized. In a library-based approach, an overall model is assembled from pre-characterized component

data [5, 6, 7]. This enables rapid exploration but does not accurately account for all glue logic and implementation-level optimizations in a combined architecture. In learning-based approaches, a low-level implementation is simulated in a sampling fashion to derive a regression-based model for a complete processor or each macro-block [8, 9, 10]. Such approaches can accurately reflect the behavior of the final implementation, but require a potentially lengthy, one-time training process for each new architecture. In all cases, existing approaches require simulation at the RTL or micro-architecture level to extract internal signal information driving the generated models. There are several approaches that drive a learned hardware model using state information extracted from a functional simulation [2, 3, 4]. However, such state-based models only support capturing coarse-grain hardware transitions between different operation modes. By contrast, we aim to drive a fine-grained, data-dependent model directly from a functional simulation. Our approach supports both library- and learning-based methods, where our focus is on learning-based generation of lightweight implementation-level representations of complete hardware processors. A key concern in learning-based methods is managing model complexities without sacrificing accuracy. Existing approaches rely on sampling a subset of key signals or state variables that are identified either manually or in a trial-and-error process [8, 10]. By contrast, we automatically decompose a full power model into several simpler models based on information about cycle-specific resource utilization, which results in better accuracy while reducing learning and estimation overhead.

For software running on processors, so-called source-level or host-compiled modeling approaches have recently emerged as an alternative to micro-architecture or instruction-set simulation. In such approaches, a source or IR model of the application is statically back-annotated with timing and energy estimates extracted from low-level simulations [11, 14]. To capture control-flow dependent effects, such back-annotation is typically performed at the basic block level. Our proposed approach is motivated by host-compiled software modeling. In contrast to existing host-compiled solutions, however, our custom hardware models are also aimed at accurately capturing data-dependent power effects. Instead of back-annotating static per block estimates, we dynamically annotate the functional simulation with cycle-level, data-dependent metrics in an online or offline fashion.

B. Back-Annotation Flow

Figure 1 shows an overview of our back-annotation driven hardware power and performance modeling flow. A given behavioral hardware model is synthesized down to an RTL description using a standard high-level synthesis process. In the process, we extract the intermediate representation of the design generated by the HLS tool after front-end optimizations. The proposed flow integrates a dynamic, switching activity-based timing and power model into the IR to generate the back-annotated hardware model. Working at the IR level allows us to accurately reflect source-level optimizations, such as bit width reductions that affect tracking of internal signals in the synthesized RTL datapath. At the same time, the IR is extracted in C/C++ form before back-end synthesis in the HLS tool, i.e. it remains at a fast functional level.

In addition to the IR, we extract back-end scheduling and binding information from the HLS flow. A timing back-annotation process uses this micro-architecture mapping data to capture and trace cycle-accurate performance and switching activity of the hardware at the IR level. Power model synthesis then utilizes switching activity traces generated from the timing model together with gate-level power traces obtained from the synthesized hardware implementation to learn a power model. A full power model is thereby decomposed into several simpler models using scheduling and binding information. Each decomposed power model is futher simplified using a decision tree-based feature selection to reduce the amount of switching information that needs to be collected. The simpilifed power models are trained with gate-level power traces and corresponding switching activity traces for a given set of training inputs. Finally, the trained power models are integrated with the already annotated timing model to complete the power back-annotation process. In the process, unnecessary activity tracing not utilized after feature selection is removed. This final step creates the back-annotated hardware model.

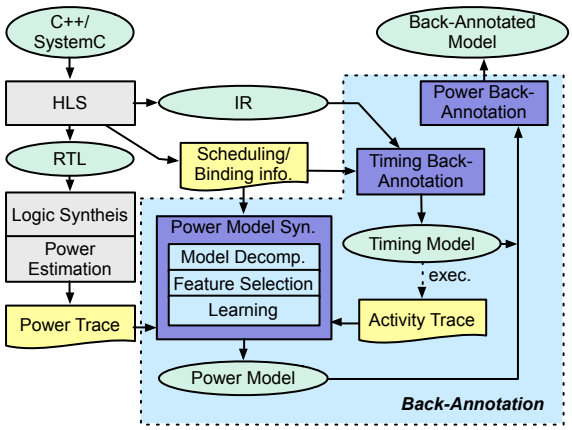


Fig. 1. Overview of hardware back-annotation flow.

II. Back-Annotation

In the following, we describe the timing and power back-annotation process through which the intermediate representations of the hardware behavior is gradually refined into a back-annotated hardware model.

A. Timing Back-Annotation

Figure 2 shows the HLS-driven timing back-annotation flow, accompanied by representative models and code snippets at various stages. Behavioral C++/SystemC code is first translated into an IR by the HLS front-end process in which control and datapath optimizations, such as bit width or operator strength reductions, loop unrolling, if-conversion and function inlining are performed. The HLS flow then converts the generated IR into a CDFG, and resource allocation, scheduling and binding steps assign the control steps and hardware resources for each node. In the HLS tool, this abstracted micro-architecture information is internally represented as a FSMD model before generating the final RTL.

1) Cycle-level signal tracing: We extract both the IR-level code as well as FSMD-level scheduling and binding information from the HLS tool. By back-annotating the abstracted micro-architecture information into the IR, we are able to trace cycle-accurate switching activity of each datapath resource at the IR level. We first extract information about the mapping of IR operations into control steps and datapath resources from

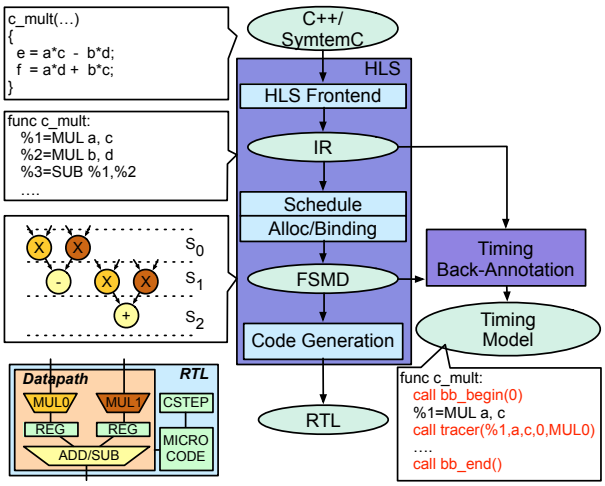


Fig. 2. Timing back-annotation process.

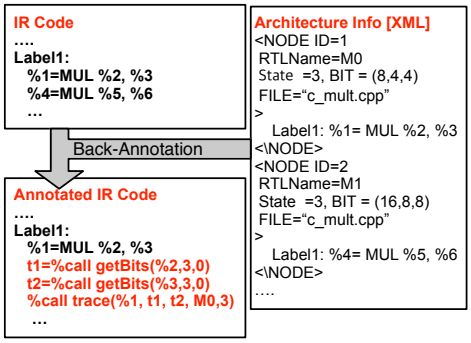


Fig. 3. Signal extraction process.

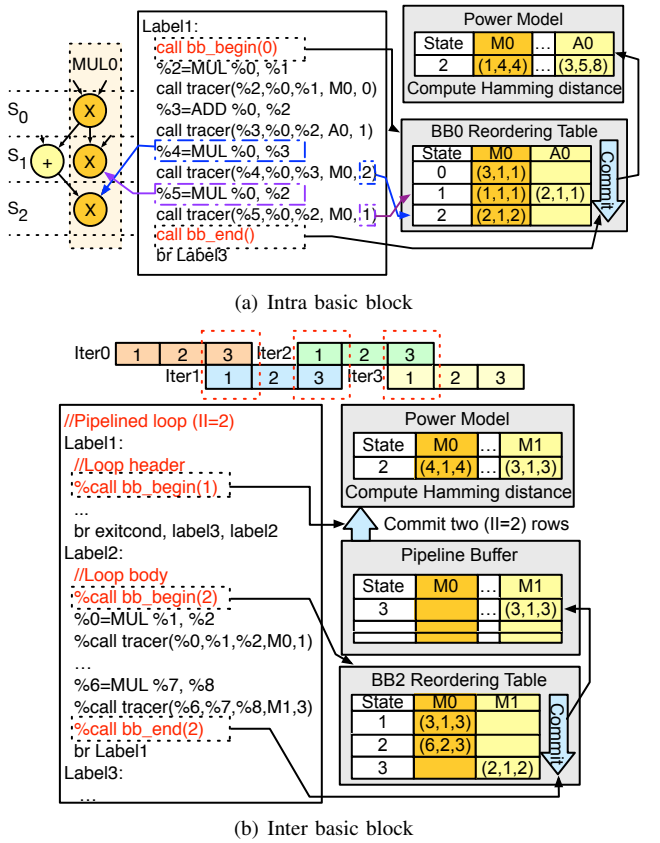


Fig. 4. Signal trace rearrangement.

the FSMD model. This information allows us to track cycle-by-cycle activity while taking into account resource sharing and other back-end synthesis optimizations. We capture the flow of data and associated switching activity by tracing the operands and results of each IR operation. To later map data activity into signal transitions of actual hardware resources, we further include resource binding information in the captured traces.

Figure 3 shows code snippets for the signal extraction and tracing process. The mapping information is provided by the HLS tool in the form of an FSMD architecture file that stores each operation node's scheduling, binding, and bit width information. We back-annotate the IR code with calls to a *trace()* function, which stores the operands and results of each IR operation together with the resource scheduling and binding information to later compute the switching activity. To take bit width optimizations into account, an additional *getBits()* function is annotated to extract the actual number of bits utilized in the hardware.

2) Switching activity computation: The synthesized hardware implementation will generally exploit operation-level parallelism and scheduling flexibility to achieve maximum performance under given resource or timing constraints. As a result, operators in the IR are not necessarily simulated in the same order in which they execute in the final hardware. Figure 4(a) and 4(b) show such intra- and inter-block level out of order execution scenarios, respectively.

In order to rearrange out-of-order execution traces captured in the IR simulation into in-order traces for hardware estimation, we perform an online reordering of traced information. As shown in Figure 4(a), the execution order of two operators in the same basic block can be reversed during back-end HLS scheduling if there is no dependency between the operations. We associate a reordering table with each basic block. In these tables, row and column tags indicate the control states and utilized resource IDs in the current basic block, respectively. The tracing function stores operands and results in the table based on annotated control step and resource data. At the end of each basic block, an additionally annotated function is called to sequentially compute the Hamming distances of all signals toggling in each control step. This switching activity information is then committed to either a tracing file or the final power model. In addition, for performance estimation, a global cycle counter is increased by the number of cycles spent in the block.

As shown in Figure 4(b), the execution of basic blocks can be overlapped in case of pipelined hardware loops. This results in some operators in the second iteration to be executed before the last operator in the first iteration. To accurately account for such effects, we introduce an additional pipeline buffer that retains the signal traces of previous iterations to restore out-of-order executions across basic blocks. Location and parameters of pipelined blocks are extracted from the HLS tool together with other scheduling and binding information. When first entering the header block of a pipelined loop, an initial pipeline buffer data structure is created. If a pipelined execution is then detected at the end of a loop body block, the entries of the reordering table are first committed to the separate pipeline buffer, and before the start of each loop iteration, all completed control steps, i.e. entries corresponding to the loop initiation interval (II) are committed from the pipeline buffer to the

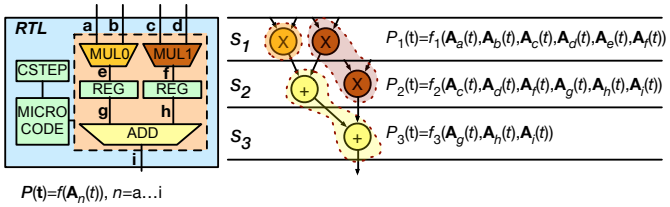


Fig. 5. Example of structural power model decomposition.

tracing file or power model. With each such iteration, entries of the loop body reordering table are sequentially committed to the pipeline buffer where they will be added to the remaining buffer contents, which will contain uncommitted signal data from previous iterations. After end of execution of a loop, all remaining entries in the pipeline buffer are committed and the buffer is deleted to emulate flushing of the pipeline. Overall, this approach allows us to accurately trace the signal transitions of hardware resources without the need for a lockstep pipeline simulation.

B. Power Back-Annotation

After collecting tracing information from the back-annotated timing model using simulations of a training set, power models are synthesized in an offline, one-time learning process described in subsequent sections. For online estimation using actual input data, the power back-annotation process integrates the trained power model with the existing switching activity annotations to create the final hardware model. Synthesized power models are thereby able to compute cycle-level and data-dependent power consumption estimates from the captured activity traces in an online or offline fashion.

For online power estimation, existing trace function calls inserted during timing back-annotation are augmented to replace the tracing of computed switching activity with calls to a regression model library. At the start of hardware simulation, pre-compiled power model parameters, coefficients and data structures are loaded into matching regression models for each control step. At runtime, the power model corresponding to the current control step then estimates its power consumption from the dynamically computed switching activities.

As part of the power back-annotation process, unnecessary signal tracing calls inserted during timing back-annotation are removed. For timing back-annotation, all arithmetic operators in the datapath are traced, which can negatively affect the simulation speed. After feature selection and decomposition of power models (as described in the following sections), the power models are simplified by reducing the number of signals needed to model power consumption. Discarded signals are removed from the back-annotation to reduce the runtime overhead of extracting switching activity information.

III. Power Model Synthesis

We utilize machine learning techniques to synthesize a power model from activity data collected during a one-time, offline training phase. As indicated in Figure 1, this process consists of three steps: decomposing power models, simplifying power models with feature selection, and learning to train and synthesize final power models.

A. Structural Power Model Decomposition

Existing approaches for power estimation at the RTL or micro-architecture level mostly rely on linear functions to model the relationship between switching activity and power consumption of a hardware module. Given switching vectors $\mathbf{A}(t)$ of internal or external signals, the power consumption $(P(t))$ can be modeled as a linear equation $P(t) = \mathbf{C} \cdot \mathbf{A}(t)$, where $\mathbf{C}$ is a coefficient vector. To simplify such a model, related signals, e.g. of busses can be grouped and switching activities modeled as the Hamming distances within the group. To find the model coefficients, linear least squares regression over a sampled set of training vectors has been widely employed. In general, a linear least squares regressor is not well suited for handling of high dimensional vectors, and a large number of training vectors is required to avoid overfitting. Feature selection of a key subset of signals can reduce the model dimensions, but this may result in a loss of accuracy.

As an alternative to traditional feature selection, we introduce an approach for reducing power model dimensions that exploits available scheduling and binding information to identify and remove unnecessary signals. We thereby decompose a global power model for a complete hardware module into separate models for each control step. For each FSMD state, the power consumption only depends on switching activities of operators scheduled in this step, which can be modeled with a subset of all signals. We illustrate this with the help of a small example. Figure 5 shows a simple micro-architecture in which three resources (*MUL0*, *MUL1* and *ADD*) are allocated. The power consumption of the complete hardware processor can be estimated as a function of the switching vectors $\mathbf{A}_n(t), n = \mathrm{a} \ldots \mathrm{i}$ of the signals connecting the internal resources. By contrast, the power model $P_i(t)$ of a given control state S_i can be constructed from the much smaller subset of signals connecting the resources scheduled in the given state only. For example, the power consumption in state S_3 $(P_3(t))$ can be estimated from three signals instead of all nine. All other signals are known to not toggle during this state. As such, a power model decomposition based on structural micro-architecture information is able to reduce the complexity of the model with little to no information loss.

B. Feature Selection

Decomposition based on the FSMD information still has limitations in handling states with high resource utilization, such as pipelined states with many scheduled operators. Moreover, decomposition still requires all signals to be traced across states, which decreases simulation speed. We therfore apply additional feature selection to further reduce complexity and improve estimation latency.

As part of basic timing back-annotation, we already select only key signals to trace based on the expected power contribution of resources in the micro-architecture. The power consumption of complex units, such as adders, multipliers or registers will be much higher with larger variations than the power of simple logic units, such as multiplexers or bitwise logic operators. Hence, we only sample the signals connected to such resources. Based on the resource mapping information extracted from the FSMD, we trace input and output signals for IR operations mapped to arithmetic units. To also take registers into account, we extract the variable mapping information from the FSMD and trace any outputs of operations that store their results in registers.

To further reduce feature sets, we additionaly leverage a decision tree approach from machine learning [15]. Decision trees are well known for their ability to automatically deter-

TABLE I. BENCHMARK SUMMARY.

	Piped	States	RTL op.	IR op.	Gates	Invoc.	Cycles
GEMM	No	6	11	11	703	3,000	2,202,000
	Yes	4	20	20	964	3,000	1,308,000
DCT	No	23	88	139	7007	10,800	1,933,200
	Yes	12	62	127	6309	10,800	1,015,200
HDR	No	18	35	69	4883	1,300	1,293,500
	Yes	19	41	103	7887	1,300	1,072,500

mine relative importance of features from the training data. We apply such feature selection after model decomposition. Feature selection first trains a decision tree model, extracts the importance of the signals, and then selects the key signals that exceed a given threshold.

C. Learning

Each power model is trained from given power and activity traces using established machine learning algorithms. Activity traces contain cycle times, states and corresponding switching vectors. Power traces contain actual power measurements from an equivalent gate-level simulation for the same set of training inputs. Activity and power traces are partitioned into states and inputs based on decomposed power models in each control step. Each power model is then trained with the corresponding partitioned traces and checked for accuracy using cross-validation methods.

Power behavior of complex arithmetic units is generally not linear [6]. We thus support linear as well as non-linear regression models. Depending on hardware functionality, input data statistics and complexity of models, a non-linear machine learning model can represent the power consumption behavior better than a typical linear least squares model. However, this comes at the expense of estimation overhead. Our learning flow utilizes a cross-validation based model selection to find the best power model for given a training set. In doing so, power model synthesis trains each available learning model with the given training vectors and picks the final model according to cross-validation scores.

IV. EXPERIMENTAL RESULTS

We have implemented a fully automated realization of our back-annotation driven power and performance hardware modeling flow utilizing the Vivado HLS [16] engine (based on an LLVM IR) and the scikit-learn [17] machine learning library. We applied this flow to generate models for pipelined and non-pipelined hardware designs of a 6x6 general matrix-matrix multiplication (GEMM), a 2D discrete cosine transform (DCT) and a weight computation block of a high dynamic range (HDR) imaging application [18]. Hardware designs were synthesized using Synopsys Design Compiler with the Nangate 45nm Open Cell Library [19] at 200Mhz clock frequency. GEMM, DCT and HDR designs were simulated with 3000 random test matrices, a 640x320 24-bit RGB image and a 200x100 24-bit RGB image, respectively. Gate-level power was estimated using Synopsys PrimeTime PX with VCD files generated from full gate-level simulation. All experiments were performed on a quad-core Intel i7 workstation running at 3.5 GHz. Table I summarizes benchmarks and synthesis results including key IR operators and (shared) RTL resources selected for initial back-annotation and tracing.

To learn power models, we used training sets consisting of separate matrices and images with 300 GEMM, DCT and HDR invocations each. In each case, we were able to synthesize power models within 10 minutes including trace generation. Depending on the trace length, model synthesis takes between 80 and 420 seconds for one-time gate-level simulation plus 20-120 seconds for total training time. We employed a 6-fold cross validation based model selection, where decomposed power models for each state are selected among a linear least squares, a linear Bayes ridged, a non-linear decision tree, and a non-linear gradient boosting regressor. For all cases and states, a non-linear decision tree or gradient boosting regressor was selected as the final power model. During decision tree based feature selection, signals of RTL resources whose aggregated importance was larger than the average resource importance were chosen. Results of this feature selection are summarized in Table II.

TABLE II. RESULTS OF DECISION TREE BASED FEATURE SELECTION.

	GEMM		DCT		HDR	
Piped	No	Yes	No	Yes	No	Yes
RTL op.	5 (-55%)	4 (-80%)	9 (-90%)	13 (-79%)	9 (-74%)	9 (-78%)
IR op.	5 (-55%)	4 (-80%)	28 (-80%)	37 (-71%)	26 (-62%)	41 (-60%)

TABLE III. SIMULATION SPEED [CYCLES/S].

	Piped	C	BA-L	BA-NL	$\mathrm{BA_S}$-NL	RTL	Gate
MAT	No	146.8M	4.87M	1.31M	1.97M	48.9K	612
	Yes	87.2M	0.64M	0.42M	1.03M	30.4K	363
DCT	No	32.2M	1.19M	1.11M	2.02M	16.1K	413
	Yes	16.9M	0.36M	0.30M	0.85M	6.8K	188
HDR	No	64.7M	2.26M	1.22M	1.47M	23.5K	276
	Yes	53.6M	1.85M	0.85M	1.11M	21.5K	199
Avg.		66.9M	1.86M	0.87M	1.41M	24.53K	342

Table III shows the simulation speeds of back-annotated (BA) models as compared to those of a pure source-level, RTL or gate-level simulation. We compare between back-annotated models with a traditional least squares regressor (BA-L), the non-linear result after model selection (BA-NL), and the final simplified model including decision tree based feature selection ($\mathrm{BA_S}$-NL). Least squares regressors were directly implemented in C++ while non-linear models were integrated as calls to the scikit-learn library using Python bindings, which results in additional simulation overhead. We can observe that decision tree based feature selection improves simulation throughput by a factor of 1.6 on average. Compared to a pure source-level simulation, the back-annotated model ($\mathrm{BA_S}$-NL) is on average 47 times slower. It is, however nearly 4100 and 57 times faster than a gate-level or RTL power simulation, respectively.

Figures 6 and 7 compare the accuracy of the power models against a gate-level power simulation. We measure both cycle-by-cycle as well as data-dependent invocation-by-invocation mean absolute error (MAE) normalized against average power. For the non-linear models, we separately applied structural decomposition (BA-NLD) and structural decomposition including decision tree based feature selection ($\mathrm{BA_S}$-NLD).

On a cycle-by-cycle level (Figure 6), selection of non-linear models results in an up to 3% better MAE than traditional least squares models used in literature. In the GEMM case, the least squares models are comparable since the complexity of the datapath and hence power models is low (11 and 20 non-pipelined and pipelined resources, respectively). In the HDR case, least squares regression was unable to find a stable model. We can further observe that, in all cases, decomposition improves accuracy. Note that decomposition does not incur any simulation overhead, i.e. these accuracy gains do not come with a reduction in simulation speed. Overall, structural decomposition reduces cycle-by-cycle MAE by up to 23% compared to basic NL models. After finally applying decision

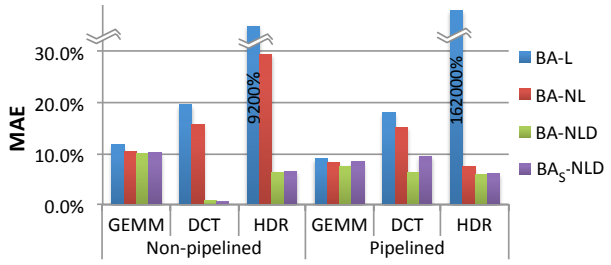


Fig. 6. Cycle-by-cycle power model accuracy.

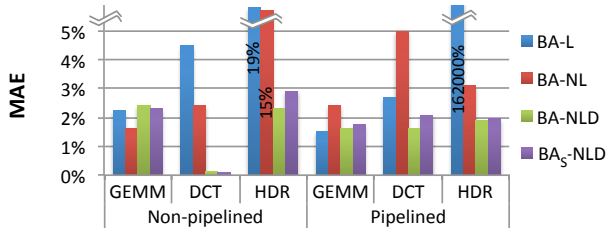


Fig. 7. Accuracy of modeling invocation-by-invocation average power.

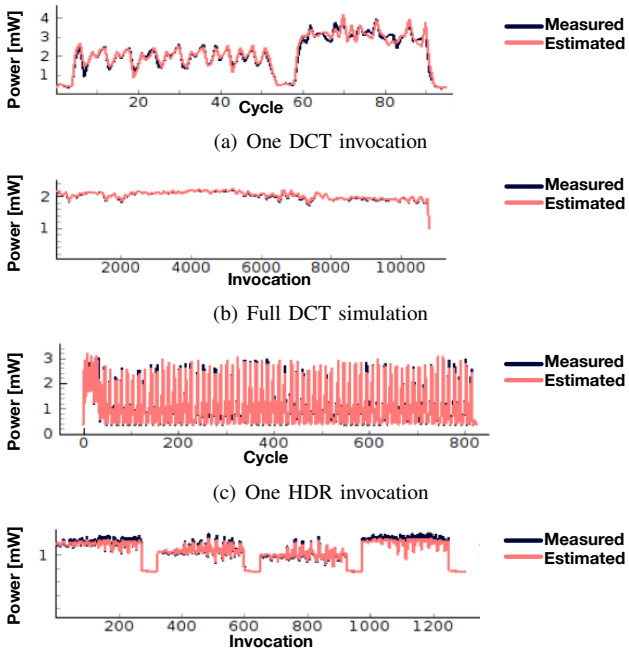
tree based feature selection, simulation speed is significantly improved, but additional inaccuracies can be introduced. The MAE of the pipelined DCT models is increased by 3%. In all other cases, MAE increases by less than 1%.

In terms of invocation-by-invocation error (Figure 7), results are similar. Decomposition can significantly improve accuracy, while decision tree based feature selection tends to incur an accuracy loss. Overall, our final models (BA$_S$-NLD) estimate cycle- and invocation-level power consumption to within 10% and 3%, respectively, compared to gate-level power results. In all cases, average errors across the whole simulation are below 1%.

Finally, Figure 8 shows the cycle-by-cycle and invocation-by-invocation profiles of estimated versus measured power waveforms for the pipelined DCT and HDR designs. As the profiles show, our back-annotated models are 100% timing accurate and can accurately track cycle-level power behavior within each invocation as well as data-dependent effects across different invocations of the same design.

V. Summary and Conclusions

In this paper, we presented a novel approach for generating fast functional hardware models back-annotated with cycle-accurate and data-dependent power and performance estimates. Our back-annotation approach is fully automated by integrating with commercial off-the-shelf tools for custom hardware synthesized by high level synthesis. The proposed power model synthesis flow exploits structural scheduling and binding information to generate accurate and fast power models using advanced machine learning techniques. Associated back-annotated models capture key signal transitions without detailed full micro-architecture simulation. Our flow has been evaluated on several industry-strength benchmarks and generated models. Results show that our approach is able to achieve orders of magnitude speedup compared to gate-level or RTL power simulation, all while producing fully cycle-accurate timing results and estimating power with less than 10% cycle-by-cycle and less than 1% average error. In future work, we plan to integrate such fast and accurate power and performance models with virtual platform or full-system simulators to support system-level architecture exploration.



(a) One DCT invocation

(b) Full DCT simulation

(c) One HDR invocation

(d) Full HDR simulation

Fig. 8. Power traces for pipelined designs.

VI. Acknowledgments

This work has been supported by Intel. We would like to specifically thank Kyungtae Han and Emily Shriver for their invaluable help throughout the project.

References

[1] I. Lee *et al.*, "PowerViP: SoC power estimation framework at transaction level," in *ASP-DAC*.
[2] C. Trabelsi *et al.*, "A model-driven approach for hybrid power estimation in embedded systems design," *EURASIP*, vol. 2011, no. 1, Mar. 2011.
[3] S. Schürmans *et al.*, "Creation of ESL power models for communication architectures using automatic calibration," in *DAC*, May 2013.
[4] E. Copty *et al.*, "Transaction level statistical analysis for efficient micro-architectural power and performance studies," in *DAC*, 2011.
[5] S. Ravi, A. Raghunathan, and S. Chakradhar, "Efficient RTL power estimation for large designs," in *VLSID*, 2003.
[6] A. Bogliolo, L. Benini, and G. De Micheli, "Regression-based RTL power modeling," *TODES*, vol. 5, no. 3, pp. 337–372, Jul. 2000.
[7] S. Gupta and F. Najm, "Power modeling for high-level power estimation," *TVLSI*, vol. 8, no. 1, pp. 18–29, Feb. 2000.
[8] D. Sunwoo *et al.*, "PrEsto: An FPGA-accelerated power estimation methodology for complex systems," in *FPL*, Aug. 2010.
[9] Y. Park *et al.*, "A multi-granularity power modeling methodology for embedded processors," *TVLSI*, vol. 19, no. 4, pp. 668–681, Apr. 2011.
[10] C.-W. Hsu *et al.*, "PowerDepot: integrating IP-based power modeling with ESL power analysis for multi-core SoC designs," in *DAC*, 2011.
[11] K. Grüttner *et al.*, "An ESL timing & power estimation and simulation framework for heterogeneous SoCs," in *SAMOS*, 2014.
[12] Y. S. Shao *et al.*, "Aladdin: A pre-RTL, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *ISCA*, Jun. 2014.
[13] D. Chen *et al.*, "High-Level Power Estimation and Low-Power Design Space Exploration for FPGAs," in *ASP-DAC*, Jan. 2007.
[14] S. Chakravarty, Z. Zhao, and A. Gerstlauer, "Automated, retargetable back-annotation for host compiled performance and power modeling," in *CODES+ISSS*, 2013.
[15] C. Ratanamahatana and D. Gunopulos, "Feature selection for the naive bayesian classifier using decision trees," *AAI*, vol. 17, no. 5-6, pp. 475–487, 2003.
[16] Xilinx, "Vivado high-level synthesis," http://www.xilinx.com.
[17] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *JMLR*, vol. 12, pp. 2825–2830, 2011.
[18] T. Mertens *et al.*, "Exposure fusion," in *PG*, Oct. 2007.
[19] Nangate, "Open Cell Library," http://www.nangate.com.