DELPHI: A Framework for RTL-Based Architecture Design Evaluation Using DSENT Models

Michael K. Papamichael* Cagla Cakir* Chen Sun† Chia-Hsin Owen Chen†
James C. Hoe* Ken Mai* Li-Shiuan Peh† Vladimir Stojanovic‡

*Carnegie Mellon University, †Massachusetts Institute of Technology, ‡University of California, Berkeley

***Abstract*—Computer architects are increasingly interested in evaluating their ideas at the register-transfer level (RTL) to gain more precise insights on the key characteristics (frequency, area, power) of a micro/architectural design proposal. However, the RTL synthesis process is notoriously tedious, slow, and error-prone and is often outside the area of expertise of a typical computer architect, as it requires familiarity with complex CAD flows, hard-to-get tools and standard cell libraries. The effort is further multiplied when targeting multiple technology nodes and standard cell variants to study technology dependence.**

This paper presents DELPHI, a flexible, open framework that leverages the DSENT modeling engine for faster, easier, and more efficient characterization of RTL hardware designs. DELPHI first synthesizes a Verilog or VHDL RTL design (either using the industry-standard Synopsys Design Compiler tool or a combination of open-source tools) to an intermediate structural netlist. It then processes the resulting synthesized netlist to generate a technology-independent DSENT design model. This model can then be used within a modified version of the DSENT flow to perform very fast—one to two orders of magnitude faster than full RTL synthesis—estimation of hardware performance characteristics, such as frequency, area, and power across a variety of DSENT technology models (e.g., 65nm Bulk, 32nm SOI, 11nm Tri-Gate, etc.). In our evaluation using 26 RTL design examples, DELPHI and DSENT were consistently able to closely track and capture design trends of conventional RTL synthesis results without the associated delay and complexity. We are releasing the full DELPHI framework (including a fully open-source flow) at http://www.ece.cmu.edu/CALCM/delphi/.

I. Introduction

Computer architects have predominantly relied on software-based simulation to evaluate the performance and other qualities of design proposals. Unfortunately, simulators—even high fidelity cycle-accurate ones—typically do not capture low-level hardware implementation artifacts, such as area overhead, increase in power consumption or timing-related side-effects of micro/architectural design choices. To overcome these limitations of simulation and gain more precise insight and deeper understanding of a proposed idea, computer micro/architectural studies have increasingly incorporated register-transfer level (RTL) design investigation to complement simulation studies.

Evaluating a design or sub-component of a system as RTL models typically entails designing and developing a low-level structural hardware implementation using a Hardware Description Language (HDL), such as Verilog or VHDL, and then taking this HDL through an Electronic Design Automation (EDA) flow. Even though EDA flows consist of multiple steps and tools, architects typically only focus on the first step, *synthesis*, which takes the HDL source and implements it using a set of primitive logic standard cells. While still far from the final chip, synthesis can provide a rough—Section II elaborates on this—estimate on the quality of a design (power, area, timing).

As is true for most steps of an EDA flow, synthesis can be a tedious, time-consuming, slow, and error-prone process that requires expert knowledge and access to costly and hard-to-get tools, proprietary process design kits, and standard cell libraries. Moreover, synthesis—which can take from a few minutes to several hours depending on design size, complexity, and implementation goals/constraints—has to be repeated for each implementation variant targeting different technology nodes and standard cell libraries. Finally, because synthesis tools are designed to be chained and used in conjunction with other commercial closed-source specialized EDA tools, they maintain and operate on a low-level internal representation of a design, which hinders integration with traditional software-based simulation frameworks. This makes it very challenging to co-simulate or combine hardware RTL modules with software-based components.

This paper presents DELPHI, a flexible open framework that leverages DSENT electrical modeling for timing, area, and power [20] for faster, easier, and more efficient characterization of RTL hardware designs. Previously, DSENT could only be applied to hand-created DSENT design models in a DSENT-specific format. DELPHI enables DSENT modeling to be applied to generic RTL designs by translating a synthesized RTL-based design (in the form of a structural netlist) into a software-based technology-independent DSENT design model. This model can then be used within a modified version of the DSENT flow to perform very fast—one to two orders of magnitude faster than synthesis—estimation of hardware performance characteristics, such as frequency, area, and power.

Compared to traditional EDA flows, DELPHI is a simpler and faster alternative that captures design trends and is consistent with actual synthesis results. By utilizing DSENT design models, DELPHI allows researchers to 1) use existing or new custom-built DSENT technology models to quickly sweep over a large number of target technology nodes and

standard cell variants without having to repeat the lengthy and complex synthesis process. Conversely, researchers are also able to quickly explore how low-level technology parameter changes impact high-level characteristics of designs. The generated models seamlessly interface with the DSENT framework and facilitate integration within larger software simulation frameworks.

Paper outline. The rest of this paper is organized as follows. Section II provides background on RTL synthesis and characterization flows and known issues with relying on RTL-synthesis results. Section III offers additional background on the DSENT tool. In Section IV we present the DELPHI flow and discuss its strengths and limitations. Section V reports our evaluation methodology and experimental results. Finally, we discuss related work in Section VI and conclude in Section VII.

II. Background

Detailed evaluation of a hardware design is a multi-step process that involves taking a design through a series of Electronic Design Automation (EDA) tools. This process starts with a description of a design, typically using a Hardware Description Language (HDL), such as Verilog or VHDL, and ends with a hardware implementation, that corresponds to the detailed layout of a chip. This description has to be at a sufficiently low level, also known as register-transfer level (RTL), in order for the tools to be able to map it or "synthesize" it to hardware. As a design progresses through an EDA flow, and gets closer to the final hardware implementation, more details are incorporated, and thus the tools can do a better job of accurately estimating implementation characteristics, such as how much area it occupies, how much power it consumes, and how fast it can run.

Logic synthesis is the first step in an EDA flow and is responsible for turning an RTL description of a design into a set of fundamental logic building blocks (logic gates and registers), which, for an ASIC technology, are then mapped to a collection of standard cells. The resulting standard cell instantiations and their interconnections become what is known as a (gate-level) netlist. This synthesized netlist can vary significantly based on a number of factors, including tool quality and user-specified design goals and constraints. While this post-synthesis netlist corresponds to a functionally correct implementation of the original RTL description, it does not capture significant hardware implementation details pertaining to the physical layout of a circuit, which are determined in later steps of an EDA flow, such as "Place and Route (P&R)"[1]. As such, hardware design characterization based on synthesis output can often deviate significantly from the final implementation produced at the end of the EDA flow.

Confidence in synthesis results. Hardware designers and the EDA community are well aware of the "noisy" and speculative nature of synthesis results, as well as of the gap between post-synthesis and post-P&R results [3], [19], [8]. However, in the architecture community, it is quite common for researchers to place a lot of trust in synthesis results and treat them as the "ground truth". Moreover, given the complex nature of modern synthesis tools and EDA flows, it is not uncommon for non-experts to misconfigure or misuse synthesis tools, which only exacerbates the observed variance and unrepresentative nature of synthesis results.

[1]P&R refers to the EDA process of physically laying out a chip that includes finding a valid placement of its standard cells, creating a network of wires to connect them together, taking care of power distribution, and creating a clock network.

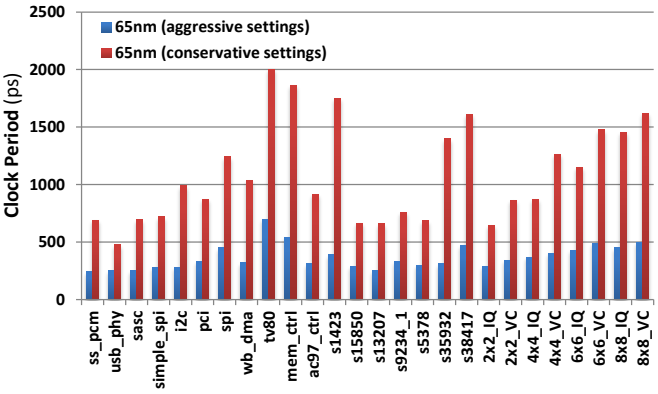

Fig. 1. Clock period with aggressive vs. conservative synthesis settings.

Below we list the most important factors that can cause variance or introduce "noise" in synthesis results, along with some illustrative examples.

- **Lack of physical layout information.** Synthesis tools treat a design at an abstract level as a collection of interconnected standard cells. Layout and other implementation details are determined at later stages of the EDA flow. As such, synthesis tools either completely ignore layout artifacts (e.g., ignore wire length and assume an ideal clock network) or use simplified models to estimate layout effects. In practice, synthesis results usually tend to be "optimistic" and it is not uncommon to see them deviate by 30%-40% compared to post-layout results. As an example, in a recent ASIC design effort within our group, our finalized design could only achieve a clock frequency that was 38% lower than the synthesis-based estimates.
- **Optimization goals, constraints, and settings.** Synthesis is an iterative process that is guided by user-specified optimization goals (e.g., optimize for area), constraints (e.g., run at 2GHz), and many other settings (e.g., effort level, driving/load assumptions for circuit inputs/outputs, etc.), which can all significantly affect the characteristics and performance of the resulting netlist (e.g., different implementations of design subcomponents, standard cell sizing, buffer insertion, register retiming, etc.). To illustrate this point, Figure 1 compares the range of reported minimum clock period for a variety of RTL designs, which are synthesized targeting the same commercial 65nm standard cell library; for each design, the differences between the red and blue bars arise purely from using different synthesis optimization/constraint settings.

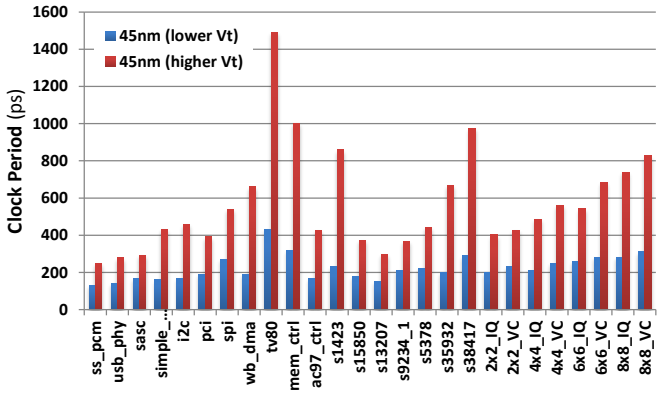

Fig. 2. Clock period with lower vs. higher Vt cells.

- **Synthesis tool features and quality.** Synthesis is a complex, highly configurable process. Depending on the selected options, different tools might use different algorithms and employ models of varying fidelity (e.g., different wire and clock tree models) during design optimization or performance estimation. Moreover, commercial tools, such as Synopsys Design Compiler, offer different variants of their tools (e.g., DC Explorer or DC Ultra) and extensions (e.g., use of Designware components, topographical technology, etc.), which can significantly affect both the generated netlist and the characterization accuracy.
- **Variance across standard cell libraries.** For a particular process technology, there are usually multiple standard cell libraries from multiple sources. The foundry will often have two distinct sets of libraries, one for internal use and one for external customers. Further, there are third-party standard cell vendors who produce libraries for a variety of process technologies, and within each of these sets of libraries there will be multiple versions (e.g., low power, high performance, compact area). As a result, even within the realm of a single process technology, synthesis results can vary dramatically. To illustrate this point, Figure 2 shows the minimum clock period achieved by a variety of designs, all targeting the same 45nm process, but using cell variants with a different threshold voltage (Vt).

In summary, when studying an RTL design at the synthesis level, it is important to keep in mind the various factors that can cause inaccuracies and variance. One must be knowledgeable and diligent about properly setting the many configuration options pertaining to synthesis. It should be noted that while synthesis results are not to be taken at face value, they are still very useful for performing first-order characterizations of designs and help guide the RTL development and optimization process.

III. The DSENT Tool

DSENT (Design Space Exploration for Network Tool) is an open-source tool developed for rapid design space exploration of photonic and conventional electrical Networks-on-Chip (NoCs), which was released in 2012 [14]. DSENT can be either used standalone as a dedicated NoC evaluation tool or it may be integrated within an architectural simulator for interconnect modeling [9]. In DELPHI, we take advantage of DSENT's design characterization technology and generalize it to support arbitrary RTL design inputs.

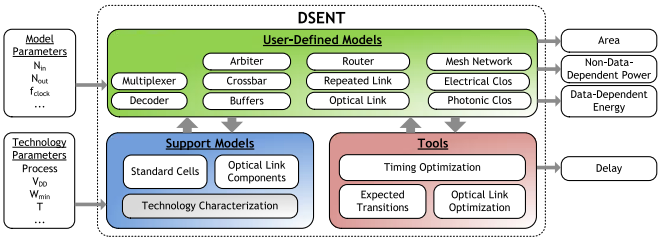

Fig. 3. DSENT internal hierarchy (from [20]).

DSENT is written in C++ and is internally organized as three distinct parts shown in Figure 3 (from [20]) and described below:

- **User-defined design models** serve as the "front-end" of DSENT that most users interact with. Within the context of NoC studies, this "front-end" contains a hierarchically organized set of parameterized building blocks that can be easily combined to assemble and experiment with a wide range of on-chip networks. In general, users that wish to extend DSENT's design library with their own custom-defined models are free to develop their own models in C++ or modify DSENT's pre-existing design models.
- **Support technology models** rely on a set of technology parameters to provide the fundamental building blocks that are used to implement (either directly or indirectly) all user-defined models. These building blocks come in the form of standard cells and optical components, whose characteristics are shaped based on a supplied technology model. The technology models used by DSENT aim at capturing the major characteristics of deep sub-100nm technologies based on a minimal set of technology parameters. While DSENT already includes technology models for 45nm, 32nm, 22nm, and 11nm technology nodes, the simple nature of the models allows users to define and calibrate their own technology models based on ITRS [6] data, SPICE models or actual process design kits.
- **Tools** provided by DSENT include a timing analysis and optimization tool, as well as infrastructure for capturing and propagating circuit switching activity information that is used to obtain accurate power estimates. Tools and support technology models form the "back-end" of DSENT, which is responsible for estimating power and timing of a design.

Expanding DSENT's front-end. DSENT's "back-end" circuit modeling engine is 1) fast, capable of characterizing a circuit in matter of seconds; 2) high fidelity, as it performs modeling at the standard cell level; and 3) flexible, allowing targeting different technology nodes through the use of simple technology models. However, this powerful back-end is limited by the library of available NoC-specific user-defined models.

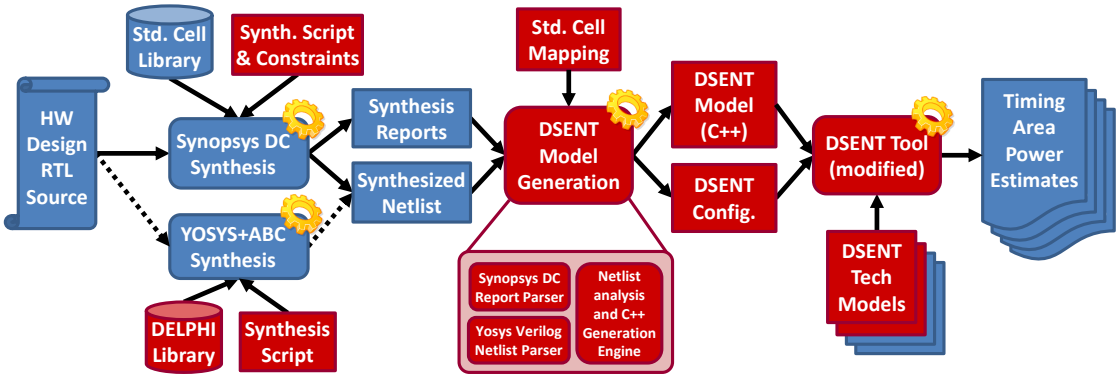

Fig. 4. The DELPHI flow. Parts shown in red were developed or modified in support of the DELPHI flow, or produced by the DELPHI flow.

Users that want to characterize their own (non-NoC) arbitrary hardware designs have to write C++ from the ground up to build new design models. This can be a tedious process and essentially resembles performing "synthesis by hand". To bridge this gap, the subject of this paper, DELPHI, provides an automated flow, that can take RTL descriptions of arbitrary hardware designs and generate a DSENT-compatible design model.

IV. DELPHI

DELPHI consists of a set of tools aimed at simplifying and accelerating hardware design characterization (determining area, clock frequency, and power). Like conventional RTL synthesis, DELPHI starts from an RTL description of a design. This RTL description is then processed through a conventional synthesis tool (e.g., Synopsys Design Compiler) to produce a netlist targeting a particular standard cell library. This netlist along with other design information is then processed in an automated manner to produce a technology-independent representation of the original design, in the form of a DSENT design model. A lightly modified version of DSENT can then use this design model to perform very fast power, area, and frequency estimations across multiple technology models.

As is the case with traditional synthesis tools, DELPHI is useful for performing coarse characterizations of RTL designs and obtaining first-order power, area, and timing results. Despite their approximate nature, DELPHI, as well as DSENT, retain and analyze a design at a low structural level. This offers higher fidelity compared to other commonly-used architectural modeling tools (e.g., [10], [15]) and allows for capturing more subtle design trends as well as identifying more specific potential areas for improvement (e.g., identify critical path in a design).

Using DELPHI. DELPHI includes all of the necessary scripts and tools required to take a design through all of the steps of the flow summarized above, including synthesizing RTL using commercial or open-source tools, generating DSENT design models, and running DSENT. From a user-perspective, DELPHI can be used in the form of a command-line utility. The user only needs to specify a minimal set of details about the RTL module to be processed (such as the name of the top-level module and the name of clock and reset ports) and, if not using the open-source flow, provide details about the particular commercial standard cell library in use.

A. The DELPHI Flow

Internally, the DELPHI flow consists of a series of steps, shown in Figure 4 and outlined below. Parts of the flow that we developed (or extended) in support of the DELPHI flow are colored in red.

Synthesis. As a starting step towards importing a hardware design, DELPHI takes the RTL design through a customized synthesis flow that ensures the resulting netlist is compatible with later steps. DELPHI supports the commercial industry-standard Synopsys Design Compiler (DC) tool, as well as a combination of open-source tools. When working with Synopsys DC, the user can synthesize a design either targeting a commercial standard cell library or using one of the freely available libraries, such as FreePDK [16] or the cell libraries provided by SPORT Lab [21]. DELPHI generates custom scripts that instruct Synopsys DC to only use the subset of standard cells available in DSENT and to generate a set of design reports that capture essential information about the design used by DELPHI.

Once synthesis is completed, DELPHI parses the synthesis output, including the generated netlist, as well as synthesis reports that include port and clock-tree information pertaining to a design, and recreates an intermediate representation of the netlist. At this point, this intermediate representation still corresponds to the standard cells belonging to the original standard cell library used during synthesis. In order for DELPHI to generate a technology-independent DSENT design model, it needs to map the vendor-specific standard cells to generic DSENT standard cells. DELPHI captures this mapping in a custom-defined specification file that assigns the various standard cell variants to their equivalent DSENT counterparts. This file also includes information on standard cell driving strength and pin mapping.

For users that do not have access to commercial synthesis tools or commercial standard cell libraries, DELPHI provides

an alternative flow based on open-source tools. In particular, the tool YOSYS [22] is used to parse the RTL of a design and then the ABC synthesis tool [2] is used with a custom-created DELPHI library and script to directly target DSENT's native standard cells[2]. Since the output of these tools is different from Synopsys DC, DELPHI includes a very basic Verilog parser that supports the subset of Verilog used in the netlists produced by these tools.

DSENT model generation. DELPHI analyzes the netlist and other information produced during synthesis to automatically generate the C++ code required to implement a DSENT design model. The creation of this model starts with the definition of input and output ports that form the model's interface. DELPHI then instantiates all of the design's standard cells and creates nets, which are used to connect all of the standard cell pins with each other and with the input and output ports of the design. If desired, DELPHI can size the instantiated cells to match the driving strengths suggested by synthesis.

At this point, the DSENT design model is structurally equivalent to the synthesized netlist. DELPHI now generates auxiliary C++ code that takes care of defining model parameters, creating a clock tree, initializing transition info for the input and output ports, as well as the clock and reset signals, and defining the events that should be monitored, gathered, and reported by DSENT. The remaining code that is generated pertains to properly propagating transition probability information in the correct order across all of the instantiated standard cells, which is necessary for performing static power analysis.

Proper propagation of switching activity information. Dynamic power is dissipated when internal nets are charged and discharged with signal transitions. Thus, to estimate power accurately, DSENT relies on annotating every input/output port and internal net of a design with signal transition probability information, which captures the likelihood of each possible signal transition (0→0, 0→1, 1→1, 1→0). Based on the type of standard cell and its input transition probability information, DSENT can calculate the transition probabilities of its outputs, which then has to be propagated to all downstream input ports of other standard cells. This propagation needs to happen in the correct order, to ensure all parts of the circuit are annotated with the correct transition probability information, which is eventually converted to a switching activity for a given frequency and used for power estimation.

To this end, a challenge in the development of DELPHI was to deduce the ordering constraints among all nodes in a netlist and then use this information to generate DSENT C++ code that will trigger the propagation of transition probability information in the correct order. DELPHI also takes special care to detect and handle sequential logic (e.g., finite state machines), which can form feedback loops that create circular dependencies. The examples that follow demonstrate the importance of updating transition probability information in the correct order. To keep matters simple in these examples, instead of looking at transition probabilities, we only focus on the state probability that a logic signal is high, which still exposes the problem at hand.

[2]This flow currently supports Verilog, but not VHDL.

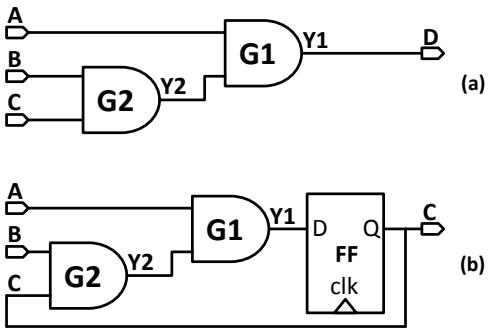

Fig. 5. Probability propagation example circuits.

Consider the very simple circuit shown in Figure 5(a) that consists of the two AND gates G1 and G2. For this example, let's assume that all inputs A, B and C have been annotated with a 50% chance of being high, i.e., P(A)=P(B)=P(C)=0.5, and internal probabilities are also initially set to 0.5, i.e., P(Y1)=P(Y2)=0.5. Our task is to find an order in which we need to process the standard cells in this circuit to correctly deduce the internal probabilities, P(Y1) and P(Y2). Since G1 depends on the output of G2, we first need to process G2 and then G1, which will set P(Y2)=0.25 and then process G1, which will set P(Y1)=0.125. Note that if we processed G1 before G2, we would end up with the wrong probability on the output of G1, i.e., P(Y1)=0.25, as we would be missing the updated probability info for P(Y2).

Now consider a slightly modified version of the previous circuit, shown in Figure 5(b), which now also includes a register FF (D Flip-Flop) that forms a feedback loop. As before, assume that all input probabilities are set to 50%, i.e., P(A)=P(B)=0.5. In this case, since G1 depends on G2 and G2 depends on the output of FF, which in turn depends on G1, it is not clear anymore in what order to propagate the state probabilities. To handle such cases, DELPHI takes a simple approach where it detects such loops and then exercises them in the correct order until the cells participating in the loop have been processed up to a number of times, determined by a user-configurable threshold[3].

To see this in action, assume that we set this threshold to five, which means that DELPHI will update the state probability of the cells that are part of this loop up to five times, respecting their ordering dependencies. Table I shows how the state probabilities of the various internal nodes evolve over each iteration, assuming we process the nodes in the order G2, G1 and FF. After five iterations, all signals, Y2, Y1, and C, have already been annotated with very low probabilities ($<$1/1000). Intuitively, this makes sense because in this simple circuit it is clear that if either A or B become low, it is then impossible for any of the internal nodes to ever become high

[3]Static switching activity propagation and estimation is a critical part of power estimation and has been studied extensively (e.g., [11]). While DELPHI's approach is simple and does not guarantee probability convergence, it is sufficient for getting first-order power estimates.

again. Note that if we did not iterate over this part of the circuit, we would have ended up with much higher, clearly non-representative state probabilities, which would in turn distort static power analysis and estimation.

Iteration#	P(Y2)	P(Y1)	P(C)
1	0.25	0.125	0.125
2	0.0625	0.03125	0.03125
3	0.015625	0.0078125	0.0078125
4	0.00390625	0.001953125	0.001953125
5	0.0009765625	0.00048828125	0.00048828125

TABLE I
STATE PROBABILITIES OVER SUCCESSIVE ITERATIONS.

Running DSENT. The C++ code generated in the previous step is now ready to be compiled along with the rest of the DSENT components to get power, area, and timing estimates. As the size of the generated code can reach several megabytes, DELPHI takes special care to optimize the generated C++ for fast compilation, which typically finishes in a matter of seconds even for large designs[4]. DELPHI also offers a special "debug" mode that produces C++ code that is slower to compile, but is much easier to read and work with for users that wish to study or modify the resulting DSENT design model.

B. Strengths of the DELPHI Approach

Leverages DSENT's speed, flexibility, and accuracy. After the initial synthesis phase, which only needs to be performed once, circuit characterization using DSENT technology models only takes seconds and can be repeated targeting different technology models without requiring additional time-consuming synthesis or recompilation of DSENT design models, as DSENT constructs its library at run-time. Moreover, new DSENT technology models are easy to define as they only require specifying a minimal number of parameters. Finally, when properly calibrated, DSENT's integrated timing, area, and power models have been shown to be accurate within 20% against SPICE simulations [20].

Harnessing the power of software. Creating a software-based representation of an RTL design opens the doors to many opportunities. Firstly, since the generated models appear as standard "user-defined" DSENT design models, they can be integrated and combined with all of DSENT's existing library of design models. Moreover, these models are portable, meaning that DSENT users can exchange and instantiate them along or inside their own DSENT models. Secondly, the entire DSENT framework can be modified to be integrated with software-based architectural simulation frameworks, as was done in [1], [13], [9].

Fast, automatic model creation. When manually building a DSENT model from scratch, users have to either assemble their designs out of library components or directly build them out of standard cells, which from a hardware development perspective is analogous to writing assembly code in software. This can be a tedious and time-consuming process that also implicitly places limits on the size and complexity of user-created models. By automating this process, DELPHI greatly accelerates building DSENT design models, as a single synthesis run is much faster than coding the necessary C++, and also eliminates the inevitable introduction of bugs. Moreover, given the abundance of freely available RTL hardware designs (e.g., from OpenCores [17]), it tremendously expands the potential of DSENT.

Preserves synthesis optimizations. Traditional hand-written DSENT models have to explicitly specify and fix all of the implementation details of a given hardware design. Since DELPHI relies on synthesis to build DSENT design models, it can preserve any low-level optimizations performed during synthesis. For instance, given the same RTL design, synthesis might implement subcomponents differently based on optimization goals, (e.g., switch between different adder implementations depending on power/area/timing trade-offs). By using DELPHI, optimizations performed during synthesis are preserved and can be carried onto and modeled within DSENT.

C. Limitations of the DELPHI Approach

Only as accurate as synthesis. The fidelity at which a design is modeled through DELPHI and DSENT is comparable to synthesis tools. This means that design characterization in DELPHI, like conventional RTL synthesis, ignores or makes simplifying assumptions about physical layout artifacts, such as long wire delays, congestion, and clock distribution.

Garbage in, garbage out. DELPHI and DSENT are only as good as the design model and technology model that they are used with. A low-quality poorly optimized RTL design or a wrong (or badly calibrated) DSENT technology model will obviously produce wrong or severely skewed results and could even hide or distort trends.

Minimal set of standard cells. The minimal set of cells used in DSENT's technology-independent standard cell library is not as rich as commercial standard cell libraries, which can lead to suboptimal synthesis results, especially for designs that make heavy use of cells that are not supported by DSENT. However, as we show in Section V, this constraint does not affect designs by much. Moreover, new standard cells can be easily added to DSENT and supported by the DELPHI flow to overcome this limitation.

Multiple clocks. DELPHI currently only supports designs with a single clock domain. To work around this limitation, designs with multiple clocks can be broken down across clock boundaries and modeled piecewise.

Large memories. By default, synthesis tools build memories in an RTL design as collections of latch or flip-flop standard cells. When dealing with RTL designs that contain large memories, designers will sometimes use proprietary vendor-provided tools, called "memory compilers", to generate

[4]This was not a trivial task, as our initial implementations generated C++ code that could take up large amounts of memory and tens of minutes to compile and would even occasionally crash g++ for very large designs.

optimized memory layouts that are typically more efficient than large memories built using conventional standard cells. DELPHI only supports "synthesized" memories, i.e., built out of standard cells, as custom memories generated using memory compilers are treated as "black boxes" during synthesis and cannot be processed by DELPHI. As future work, to improve large memory modeling, we are considering extending the DELPHI flow to incorporate the use of CACTI [15] memory models.

Limited to static analysis. DELPHI and DSENT support static analysis for power estimation, which can provide satisfactory estimates based on probabilistic models of signal transitions. However, more accurate power modeling requires performing simulations that stimulate the circuit using representative inputs to capture actual switching activity information, which is then fed to commercial power analysis tools, such as PrimePower.

V. Evaluation

This section presents our evaluations, based on actual synthesis results targeting commercial standard cell libraries, as well as results obtained through DELPHI and DSENT. In the presentation below, we first validate the assumptions underlying the DELPHI approach before directly evaluating the accuracy of DELPHI estimates.

A. Methodology

All synthesis results were obtained using Synopsys Design Compiler (Version I-2013.12-SP3) targeting 32nm, 45nm, 65nm, or 130nm commercial standard cell libraries that come from three different vendors. When reporting timing results we synthesize with aggressive constraints to force synthesis to reach the highest possible operating frequency for each target design. For all other results, we synthesize and obtain power results targeting the same fixed frequency of 500MHz[5]. All power estimation results assume a switching activity factor of 0.5 across all design inputs.

For DELPHI results, we use a lightly modified version of DSENT (0.91) targeting the default 45nm, 32nm, 22nm, and 11nm technology models that come bundled with the publicly released version of DSENT, which, in our results, we denote as "DSENT_45", "DSENT_32", "DSENT_22", and "DSENT_11". Our evaluations span 26 hardware designs: 18 benchmarks from the IWLS2005 benchmark suite [5] (including 11 designs from OpenCores and 7 from ISCAS) and 6 on-chip router designs of varying size and architecture obtained through the CONNECT NoC generator [18], [12]. Within each group, benchmarks are sorted according to their size.

Finally, it is important to point out that the DELPHI results shown in this section were obtained using the publicly available DSENT "generic" technology models, which do not correspond to any of the commercial standard cell libraries used in this evaluation (and which are protected under strict NDA agreements). As such, when comparing DELPHI results with actual synthesis results, emphasis is placed on capturing the same trends, instead of matching the same absolute numbers. While we focus our evaluation on the 32nm results, the most modern technology node for which we have standard cells, we observe similar results across all technology nodes that we have access to.

[5]We pick this frequency as it was the highest frequency that could be met for all designs and for all technology nodes used in our evaluation.

B. Results

Constraining synthesis to DSENT standard cell subset. An artifact of using DELPHI is that synthesis must be constrained to target the DSENT-supported set of standard cells, typically a subset of commercial libraries. The first set of results show the effects of constraining synthesis to only using the subset of standard cells that are available in DSENT. Figures 6, 7, and 8 compare power, area, and timing results from baseline ("unconstrained") and "constrained" synthesis runs targeting a commercial 32nm process.

Overall, "constrained" synthesis can experience approximately 5%-20% quality loss in the design metrics. This is expected as, e.g., a design that could make use of a 3-input AND gate will now have to switch to chaining two AND gates to implement the same logic, which in turn can increase critical path, area, and power. As will be shown later, these minor—in light of the speculative nature of synthesis—distortions do not prevent DELPHI from capturing design trends.

Independence from standard cell library choice for DSENT model creation. DELPHI and the DSENT back-end provide technology-independent flows and modeling that are tolerant to using different RTL-synthesis standard cell library targets when originally generating DSENT design models from RTL synthesis. For this next set of results we assess this premise by synthesizing each design targeting all four commercial standard cell libraries (32nm, 45nm, 65nm, and 130nm) and then processing the synthesized netlists using the DELPHI flow to generate four separate DSENT design models. We then use these four models to target DSENT's 32nm technology model (DSENT_32) to obtain timing, area, and power estimates, shown in Figures 9, 10, and 11.

The fact that these results show low variance and behave consistently across all benchmarks and estimated metrics (timing, area, and power), demonstrates that the DELPHI flow is robust to using different RTL-synthesis standard cell libraries as a proxy for generating a DSENT design model. The extent of "technology independence" becomes more apparent if one considers that these standard cells not only span a wide range of technology nodes, but also come from three different vendors, which vary significantly in their standard cell offerings.

Capturing power, area, timing trends. After having established the validity of the DELPHI/DSENT approach in the above studies, this last triplet of Figures, 12, 13, and 14, show how well DELPHI captures timing, area, and power trends by juxtaposing results from baseline ("unconstrained") full RTL

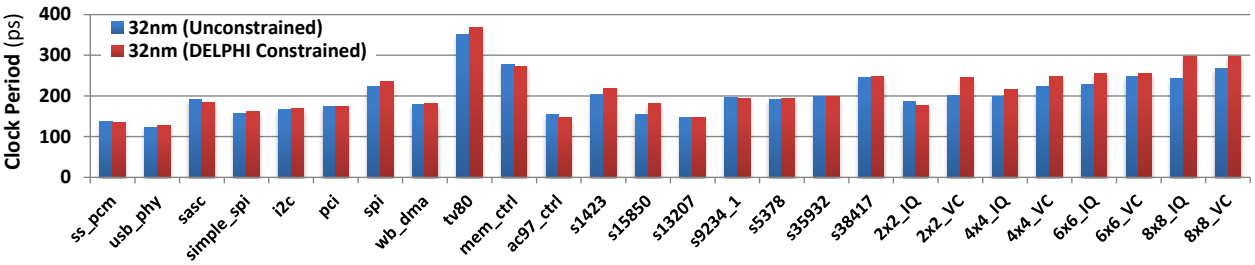

Fig. 6. Timing estimates of regular vs. DELPHI-constrained synthesis.

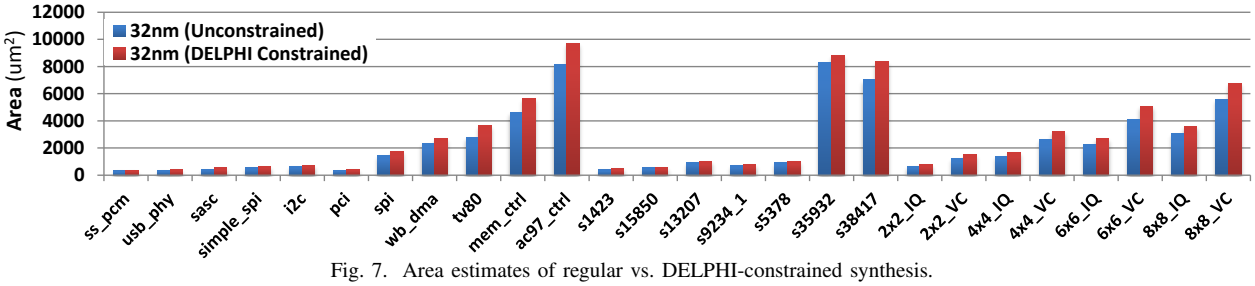

Fig. 7. Area estimates of regular vs. DELPHI-constrained synthesis.

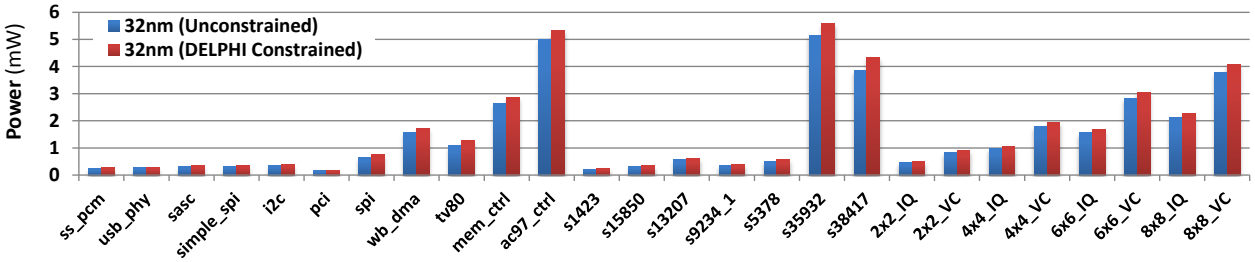

Fig. 8. Power estimates of regular vs. DELPHI-constrained synthesis.

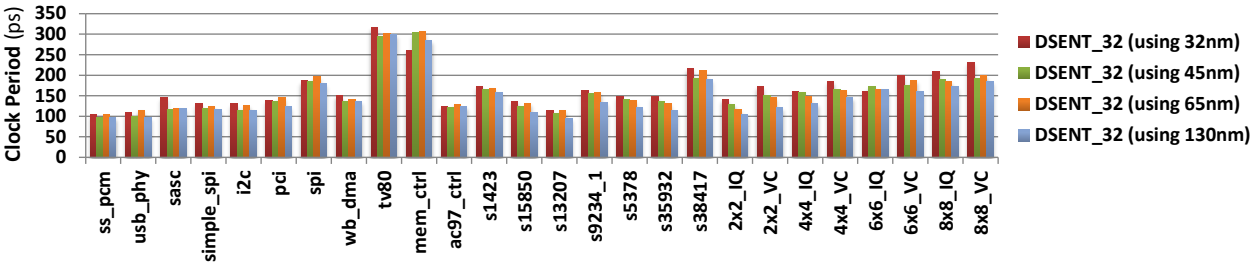

Fig. 9. Timing estimates of four DSENT design models all targeting DSENT_32, but generated using different intermediate synthesis results.

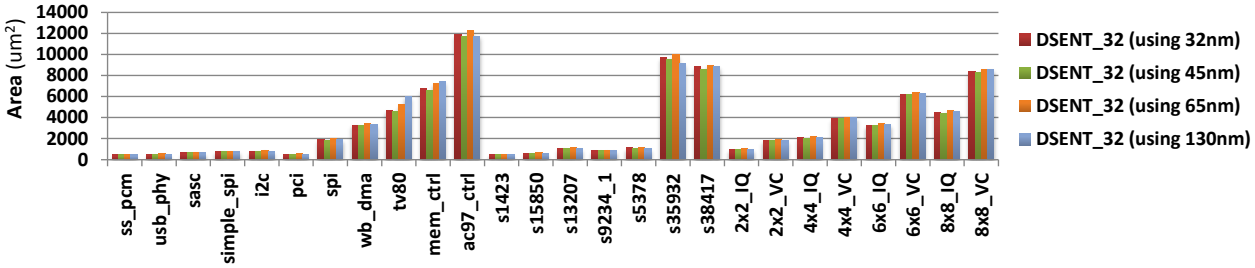

Fig. 10. Area estimates of four DSENT design models all targeting DSENT_32, but generated using different intermediate synthesis results.

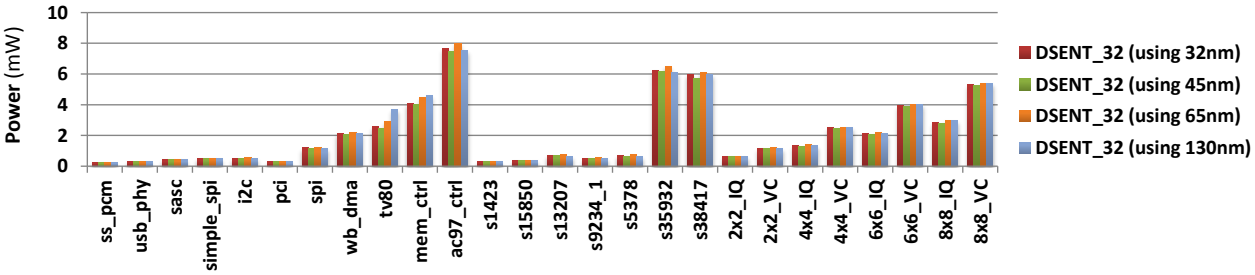

Fig. 11. Power estimates of four DSENT design models all targeting DSENT_32, but generated using different intermediate synthesis results.

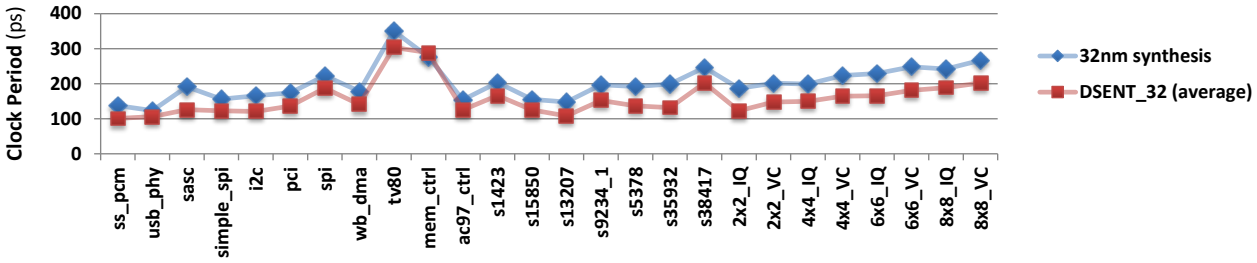

Fig. 12. Comparing trends between the average timing estimates of four DSENT models targeting DSENT_32 vs. 32nm synthesis results.

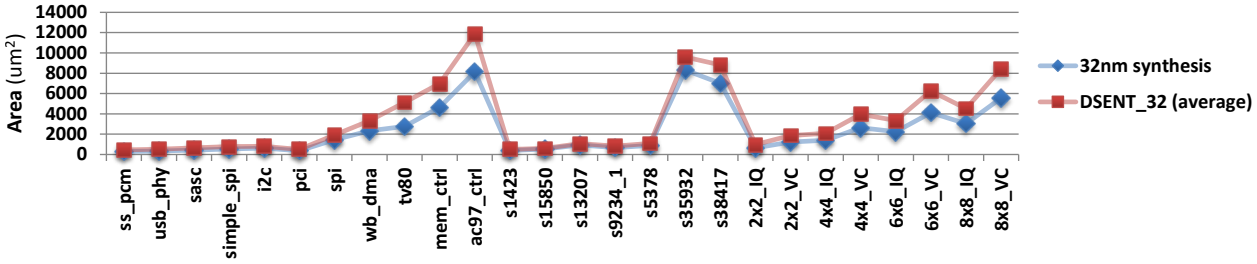

Fig. 13. Comparing trends between the average area estimates of four DSENT models targeting DSENT_32 vs. 32nm synthesis results.

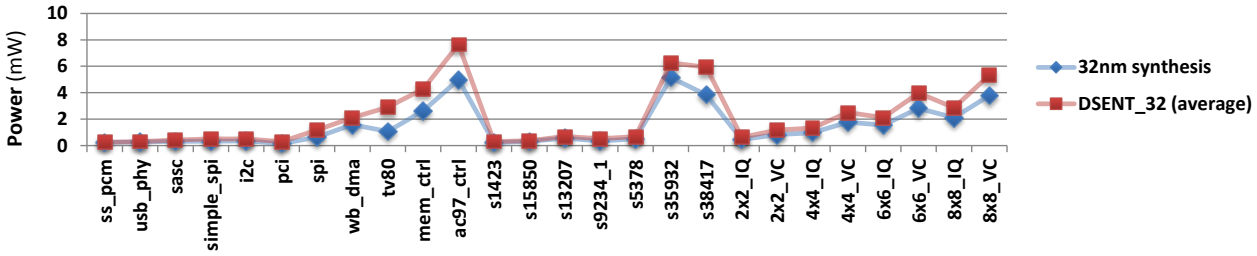

Fig. 14. Comparing trends between the average power estimates of four DSENT models targeting DSENT_32 vs. 32nm synthesis results.

synthesis targeting a commercial 32nm standard cell library against the average of the DSENT_32 estimates presented in the previous set of results. Note that the DSENT estimates behave consistently and exhibit similar trends with the actual synthesis results. As was mentioned earlier, it is important to note that the absolute values of the presented data are not meant to match, as the DSENT_32 model does not correspond and was not calibrated to match the commercial 32nm cell library used for synthesis.

DELPHI usage example. As an example usage case for DELPHI, consider a hypothetical scenario where a computer architecture researcher is interested in obtaining coarse power trends for two different Network-on-Chip (NoC) router RTL designs as technology nodes scale, including future technology nodes or nodes for which she or he does not have access to. The first router is based on a simple minimal low-performance Input-Queued (IQ) router architecture and the second router is based on a high-performance Virtual-Channel (VC) router architecture.

Assuming DELPHI is used, as a first step, these designs would have to be synthesized targeting some available standard cell library, a process that would take in the order of tens of minutes and would have to be repeated separately for each of the two router variants. The two synthesized netlists are then taken through the DELPHI flow to generate DSENT design models and compile DSENT, which typically takes less than two minutes for both models. These DSENT design models are then used to target five DSENT technology models (65nm, 45nm, 32nm, 22nm, and 11nm) and obtain power estimation results, which are shown in Table II. From these results, it is clear that the high-performance VC router becomes increasingly more attractive (from a power perspective) in future technology nodes, especially at 11nm, where the power difference compared to the IC router has decreased by an order of magnitude and has become negligible.

If the same characterization was to be performed through traditional full synthesis, it would have taken many orders of magnitude longer to perform an equivalent characterization, which would require running 10 synthesis jobs (not to mention the overhead of properly configuring the synthesis environment for five different standard cell libraries). In fact, this technology forecast study may not be possible at all using traditional synthesis, since it would require a prohibitive investment to create standard cell libraries for future non-existent technology nodes.

	Power Estimates (mW)	
Technology Model	**8x8 IQ Router**	**8x8 VC Router**
DSENT_65	11.88	21.92
DSENT_45	8.65	15.94
DSENT_32	5.49	10.10
DSENT_22	3.34	6.16
DSENT_11	1.03	1.88

TABLE II
SAMPLE NOC POWER STUDY USING THE DELPHI FLOW.

Summary. Overall, in our experiments DELPHI exhibits consistent and robust estimates across a variety of benchmarks, standard cell libraries, and technology nodes. Please keep in mind, however, that DELPHI, like post-synthesis evaluation, is meant only to expose trends and perform first-order characterization of a design. Our hope is that the results presented in this section will aid in calibrating expectations with regards to DELPHI's capabilities and the extent to which it is able to perform hardware design characterization and capture trends.

VI. RELATED WORK

To overcome complexity and speed limitations, the architecture community has a history of building and relying on models of varying fidelity to gauge the performance and characteristics of hardware components, such as processors, memories, adders, Networks-on-Chip, etc. Examples of such models include McPat [10], an integrated power, area, and timing modeling framework specific to multicore processor architectures, and CACTI [15], a tool for modeling power, area, and timing characteristics of cache memories. Other tools place particular focus on specific metrics and types of modeling, such as Wattch [4], which focuses on power estimation for microprocessors, or Orion 2.0 [7], which focuses on power and area for interconnection networks.

Compared to such models, DELPHI combined with DSENT can be thought of as a "meta-model" in the sense that it can be used to generate fast power, area, and timing estimation models based on any existing RTL-based hardware design. While previous models are either high-level, such as McPat, which introduces abstraction errors, or limited to very specific hardware subcomponents, such as CACTI, DELPHI sidesteps these issues and offers both high-fidelity and generality by operating directly at the register-transfer level.

VII. CONCLUSIONS

In this paper we presented DELPHI, a framework which leverages DSENT modeling to perform fast timing, area, and power estimation for arbitrary RTL designs. We first discuss the limitations of RTL synthesis-based estimations. We next show that the results obtained through DELPHI can capture trends and first-order effects, and are comparable to post-synthesis estimates. DELPHI combined with DSENT allows for performing rapid design space exploration across multiple technology nodes in a fraction of the time that would be required using a synthesis-based approach.

We are releasing the DELPHI framework along with any modifications to DSENT. For more information please visit http://www.ece.cmu.edu/CALCM/delphi/.

ACKNOWLEDGMENTS

Funding for this work was provided by NSF CCF-1012851. We thank the anonymous reviewers for their feedback and comments. We also thank Kaushik Vaidyanathan and Mark McCartney for the helpful discussions and sharing their EDA tool knowledge.

REFERENCES

[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *ISPASS*, April 2009.
[2] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/.
[3] D. Brand and C. Visweswariah. Inaccuracies in Power Estimation During Logic Synthesis. In *ICCAD*, November 1996.
[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *ISCA*, June 2000.
[5] Christoph Albrecht, Cadence Research Laboratories at Berkeley. IWLS 2005 Benchmarks. http://iwls.org/iwls2005/benchmarks.html.
[6] International Technology Roadmap for Semiconductors. http://www.itrs.net.
[7] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *DATE*, June 2009.
[8] E. Kounalakis. The Mythical IP Block: An Investigation of Contemporary IP Characteristics. Technical Report ICS-TR366, Institute of Computer Science, Foundation for Research and Technology - Hellas (FORTH), October 2005.
[9] G. Kurian, S. Neuman, G. Bezerra, A. Giovinazzo, S. Devadas, and J. Miller. Power Modeling and Other New Features in the Graphite Simulator. In *ISPASS*, March 2014.
[10] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *MICRO*, December 2009.
[11] R. Marculescu, D. Marculescu, and M. Pedram. Probabilistic Modeling of Dependencies During Switching Activity Analysis. *IEEE Transactions on CAD*, 17:73–83, Feb 1998.
[12] Michael K. Papamichael. The CONNECT NoC Generation Framework. http://users.ece.cmu.edu/~mpapamic/connect/.
[13] J. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal. Graphite: A Distributed Parallel Simulator for Multicores. In *HPCA*, Jan 2010.
[14] MIT. DSENT (Design Space Exploration for Network Tool). https://sites.google.com/site/mitdsent/.
[15] Naveen Muralimanohar, Rajeev Balasubramonian and Norman P. Jouppi. CACTI 6.0: A Tool to Model Large Caches. Technical Report HPL-2009-85, 2009.
[16] North Carolina State University. NCSU FreePDK. http://www.eda.ncsu.edu/wiki/FreePDK.
[17] OpenCores. OpenCores: Open Source Hardware IP-Cores. http://opencores.org/.
[18] M. K. Papamichael and J. C. Hoe. CONNECT: Re-Examining Conventional Wisdom for Designing NoCs in the Context of FPGAs. In *FPGA*, February 2012.
[19] I. Ratkovic, O. Palomar, M. Stanic, O. Unsal, A. Cristal, and M. Valero. Physical vs. Physically-Aware Estimation Flow: Case Study of Design Space Exploration of Adders. In *VLSI (ISVLSI)*, July 2014.
[20] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *NOCS*, 2012.
[21] University of Southern California. System Power Optimization and Regulation Technology (SPORT) Lab. http://sportlab.usc.edu/.
[22] C. Wolf. Yosys Open SYnthesis Suite. http://www.clifford.at/yosys/.