RT Component Sets for High Level Design Applications

Nikil D. Dutt and Pradip K. Jha

Technical Report# 94-27
August 24, 1994

Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92717-3425
Phone: (714) 856-8059
FAX: (714) 856-4056
Email: pradip@ics.uci.edu

Abstract

The system-level design process typically involves refining a design specification down to the point where each of the system's components is described as a block diagram or netlist of abstract Register-Transfer (RT) level components. In this report, we motivate the need for such a standard RT component set, and describe a library environment that supports automatic model generation, design reuse, and synthesis with technology-specific estimators. We demonstrate the efficacy of the standard RT-component set approach with experiments performed on the HLSW92 benchmarks. Our preliminary results indicate only a small overhead of about 10% in using these standard, generic components. We then describe an automatic model generation and technology projection scheme that uses fast (on-line) estimators for predicting the area and delay of generic RT components tuned to a particular technology library with an accuracy of 10%. These model generators and estimators have been integrated with a high-level synthesis system at U.C. Irvine.

Contents

List of Figures

1 Introduction

Present-day design methodologies involving schematic capture and simulation require the system designer to partition, refine and specify a design as an interconnection of components drawn from a vendor's library. These components can vary in their level of complexity from simple logic gates, to sequential components such as counters and registers, to arithmetic blocks such as ALUs, and all the way up to complex components such as CPU cores. However, the register-transfer (RT) level is a common design entry point that is supported by most of the existing CAD tools on the market. Furthermore, the RT-level has had a long history of use as a design entry point, as evidenced by the frequent use of TTL databook component names by designers, as well as in digital system design courses outlined in standard textbooks and taught at universities. We also note that most data sheets for product specifications (either being designed, or after they have been designed) are often composed of register-transfer schematics typically drawn up by system level designers.

Although RT-level components are commonly used in specifying, documenting, refining and synthesizing designs, there is a lack of standardized RT component sets that can facilitate unambiguous documentation, communication and design reuse. Only recently has an effort spearheaded by the FPGA community (LPM)[10] begun to address this issue. This is in contrast to the logic-level, where the designs can be expressed as netlists of well-understood standard components such as the equivalent 2-input NAND or NOR gate. The lack of a standardized RT-level component set is a serious roadblock to elevating the design process beyond the RT-level and will affect the capability of effectively synthesizing large-scale system designs in an efficient manner. Furthermore, the lack of a component set also prevents the capability of automatic model generation for various design tasks such as simulation, synthesis and verification – an important feature that is often crucial for supporting design reuse as well as redesign.

Component sets and libraries play a particularly important role in the context of synthesis; well defined component sets at the input and output are critical for the successful realization of any synthesis tool. We typically use *generic* components to specify the input or intermediate results of synthesis, and follow with a phase of *technology mapping* to realize the design with a set of components from a technology library [11]. For instance, logic synthesis uses generic components such as simple logic gates (e.g., AND, OR, INVERT) at the input and for intermediate synthesis steps, but the last step of logic synthesis involves technology mapping of the generic design into components drawn from a technology library (e.g., complex CMOS gates, or a different logic gate family such as NOR-NOR)[24]. Generic component sets facilitate technology independence, and allow the capture of a design in a standard form that can be retargetted to different libraries (or technologies) without changing the input description. Of course, technology independence needs to be coupled with good technology mapping strategies that can effectively map generic designs to target library components with low overhead.

Another important requirement for system-level design is the capability of specifying the design once, but using this specification to predict technology-specific design characteristics (e.g., area, speed, power) for different target implementations. System-level designers would like to perform early design space exploration by delaying binding of system-level components to a particular technology or implementation, but need the capability of rapid technology projection for different target libraries. The concepts of delayed binding, technology projection and effective estimation for system-level design cannot be performed without the support of a well defined component set and associated tools for technology mapping and prediction.

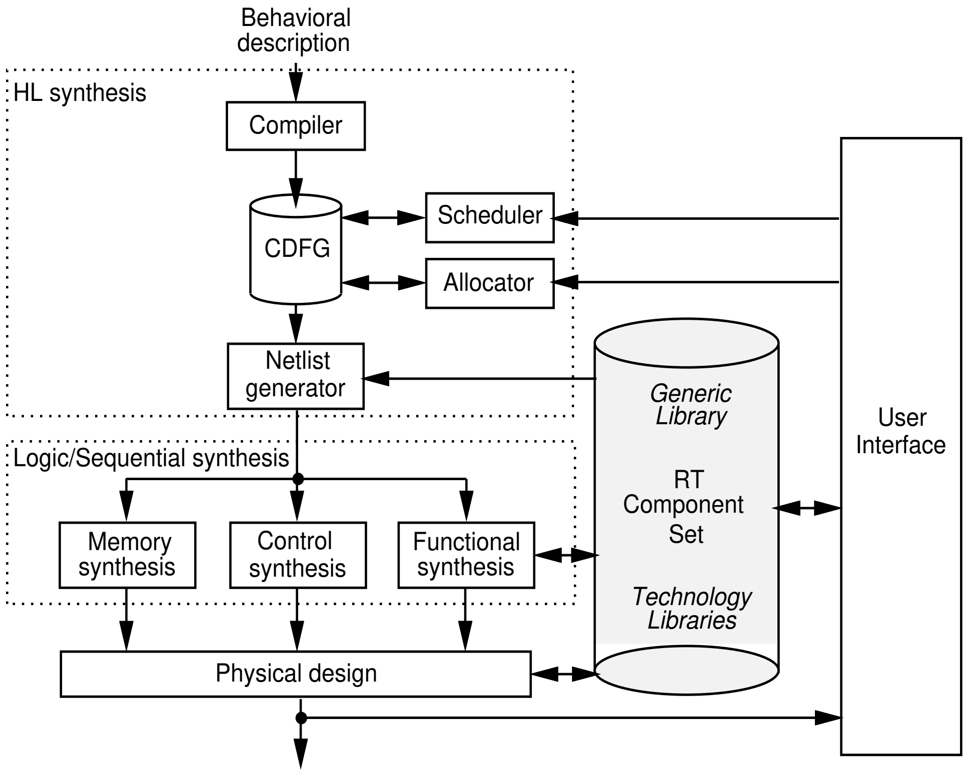

Figure 1: RT component set for high-level design

High-Level Synthesis (HLS) also relies on a library of well-defined, parameterized RT component generators to simplify the mapping of behavioral variables and operators to physical components. This mapping of the abstract design into an interconnection of RT components involves design space exploration by selecting and allocating a proper set of RT components, guided by design metrics (e.g., area and delay). The parameterized components are used as the building blocks for the tasks of allocation and binding. Each component is customized by parameterized attributes such as the required bit-width and functionality. Such a library of RT component generators provides a complete component set for HLS [9], and provides a path to physical design through logic and layout synthesis [7]. Figure 1 shows an overview of such a scheme. However, it is important to note that much of the previous efforts in HLS have focused on the tasks of scheduling and allocation (represented by the dotted box in Figure 1), and not much attention has been paid on how to support component sets for this synthesis task (the shaded box in Figure 1).

With increasing interest in high-level synthesis and higher-level design methods, the need has thus evolved for a well defined generic RT component set, together with rapid estimators that allow technology projection into different backend libraries. We briefly describe the GENUS library and discuss its comprehensiveness by comparing it with various technology libraries. We then present the results of experiments on some high-level synthesis benchmarks to evaluate the amount of overhead incurred by using generic components. Once the utility of the generic component library is established, we need to establish an automatic model generation strategy, and provide technology estimators to support high-level synthesis and design space exploration. Typical design metrics include the area, delay and power of the RT-components projected in a particular technology. We finally describe the automatic model generation strategy, as well as the technology-specific estimation technique that can support high-level synthesis.

2 Related Work

Although abstract component characterization is an important task for high-level design and synthesis, not much work has been reported on this topic. Traditional high level synthesis systems either use very abstract components with crude estimates for delay and area, or use design components drawn from a particular technology library. The direct incorporation of technology information may yield good performance estimates, but complicates the task of retargetting to new libraries and evaluating design alternatives across different technologies.

Some work on component characterization and module databases has been done at the layout and logic levels [22] [32]. VHDL [12] is a standard for design documentation and exchange, and has good constructs for describing specific libraries and component instances. However, it is unsuitable for the task of generating customized component libraries. The module generator approach to RTL component synthesis is most often observed in silicon compilers and behavioral synthesis tools. High-level synthesis systems assign operators to parameterized module generators, which are treated as architectural primitives for generating layout [6]. Some synthesis systems integrate module generators with logic synthesis tools [4]. However, no comprehensive work has been reported on characterizing RT component libraries.

The problem of area and delay estimation has been studied at several design levels and in several contexts. At the level of a complete datapath design, work has been done to predict the area-time tradeoff and this prediction has been used for component selection[13]. At the logic level, work has been done to predict the area and delay of an RT component, given its structural implementation as a netlist of logic cells or blocks [18]. LAST[21], TELE[25], [31] and [5] provide technology-driven estimates that include physical design effects (e.g., routing), given the structural netlist of the design.

Work on coupling technology independence with technology prediction has been investigated at the logic level. [30] proposes a simple technology-independent model for predicting the delay of combinational control (random) logic, given the Boolean equations for the logic (it does not deal with area estimation). Tyagi [28] uses an information-theoretic approach for estimating the area and delay of some parameterized combinational components. However, previous work has not addressed the estimation problem for the complete set of parameterized generic components including combinational and sequential components.

Kang and Szygenda address issues in automatic model generation for simulation applications [19]. They use a rule base and a model library to generate VHDL simulation models using a variety of input descriptions (truth tables, Boolean equations, schematics, etc.). Although the approach looks very promising, their work is limited to simulation model generation for logic-level schematics, and does not address behavioral model generation at the RT level.

3 High-level Design Using GENUS

The design scenario we propose consists of a well-characterized RT component library named GENUS (Figure 2). The high-level design phase (either manual design or automatic synthesis) refines the input specification into a netlist of RT components from this library. Different high-level design trajectories produce different designs consisting of netlists of RT components. Each netlist of RT components is then mapped to components from a technology library with the help of a high-level technology mapper or through logic synthesis. The components in the technology library could vary in complexity from simple 2-input logic gates to complex multifunctional RT components such as ALUs.

A standardized set of RT components is critical to the success of the high-level VLSI design process, since it provides the crucial link between the high-level design phase and the logic and layout level design phases. In order to facilitate seamless communication between these two phases, we need a standard set of components that have well-defined semantics and models to facilitate simulation, synthesis, testing and verification. The standard component set should automatically generate different models as shown in Figure 2. *Simulation models* are behavior models used to validate the input and output of each high-level design step. *Synthesis models* are used to refine the high-level design description into logic or layout-level design. *Testing models* provide test vector sets and can also provide testability functions for performing hierarchical ATPG. *Verification Models* are used to perform formal verification of the design refinement. *Technology projectors* provide rapid estimates of generic component implementation in specific technology. Such estimates are crucial for effective design space exploration and tradeoff analysis at higher levels.

However, technology independence comes at a cost – implementing a generic design with different technology libraries can result in an overhead due to the mismatches in the types of components. In the next section we briefly describe the GENUS library and discuss its comprehensiveness by comparing it with various technology libraries. We also present the results of experiments on some high-level synthesis benchmarks to evaluate the amount of overhead incurred by using generic components.

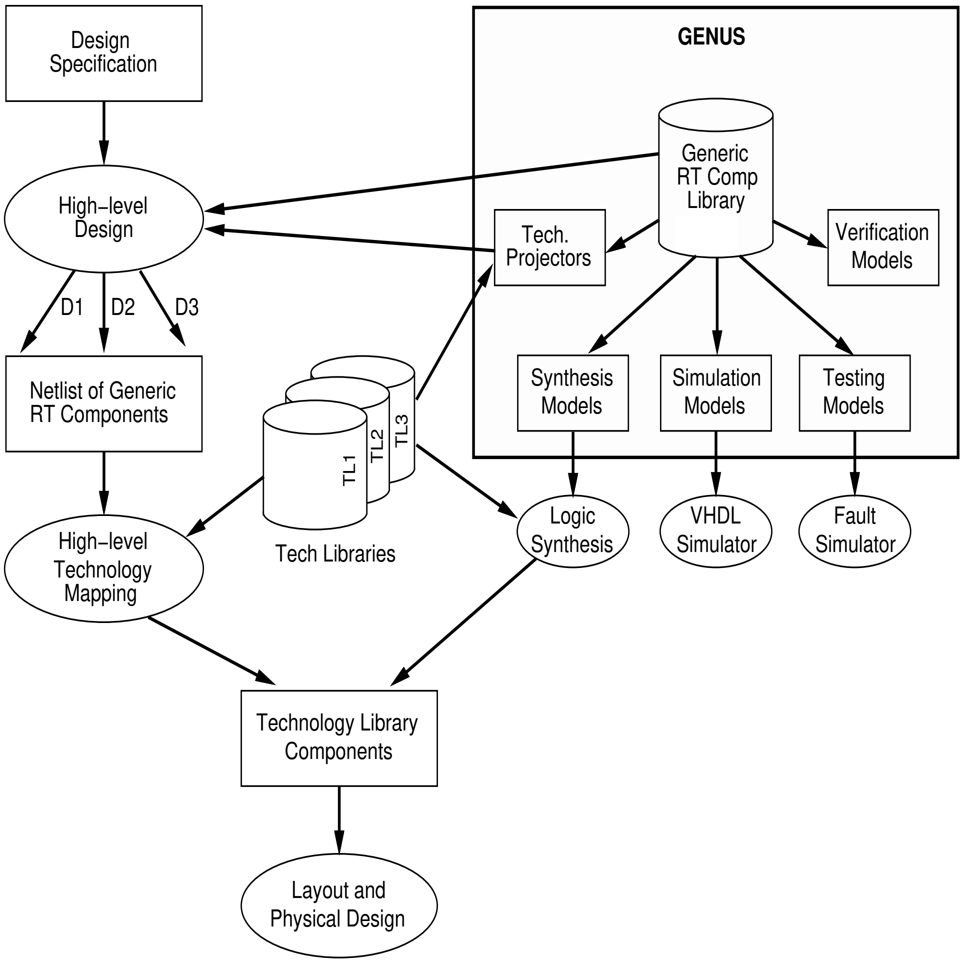

Figure 2: The design scenario

Once the utility of the generic component library is established, the high-level design phase in turn needs model generators to support high-level design tasks and technology projectors that estimate design metrics to facilitate rapid design-space exploration. Typical design metrics include the area, delay and power of the RT-components projected in a particular technology. We need to develop estimators that can factor in technology-specific information such as layout styles and physical design considerations such as placement and routing so as to provide accurate area-delay metrics specific to a particular technology. Section 5 and 6 addresses model generation and estimation respectively for the generic components.

4 The GENUS RT Library

GENUS [9][17] is a generic RT component library that consists of a set of parametrized RT component generators. The generators are defined using RT-level functionality and are grouped into classes based on functional similarity. A component instance is generated by specifying parameter for a corresponding generator. For example, an ALU generator is characterized by the following parameters: *(bit-width, set-of-functions, implementation-style)*, whereas a specific ALU instance is generated by specifying values for these parameters. The grouping of similar components into classes of generators makes the task of library management simpler and more efficient since the resulting number of generators (approximately 50) is much less than the virtually infinite number of possible component instantiations.

Besides the parameters that are used to instantiate a specific component, each generator is characterized by a well-defined interface and associated semantics. Components derived from a generator can perform a specific set of RT level functions and each generator's specification includes the set of these functions. RT-functional mappings specify the exact relationship of each output with respect to the input. GENUS follows a well-defined port naming convention – an important detail that can often make technology mapping difficult if not done consistently. Synthesis and simulation models are also available for each class of RT generator. The user interface to GENUS provides a set of routines to create, delete and query information regarding a specific component. The set of Boolean equations constitute a synthesis model that explains how the generic component can be built from logic-level primitives.

Although generic components are appealing in concept, we have to address some important issues while defining a standardized generic component set. First, how comprehensive is the generic library set? That is, how well does GENUS cover various components across different technology libraries? Second, how much penalty do we pay in using GENUS as compared to directly using the technology components? We discuss these two issues in the remainder of this section.

4.1 Generic Component Coverage

We present the results of our survey of library coverage with respect to different technology libraries that use varying layout styles for component implementation. In particular, we examined the following layout styles: Standard cell, Bitslice, Gate array and Field programmable gate array (FPGA). The first two layout styles typically result in more compact designs at the cost of longer design cycles, while the gate array and the FPGA styles provide a quick method for prototyping designs. We considered the following technology libraries in our survey: VTI Datapath Compiler

[29], Cascade Digital Library [3], Toshiba Gate Array Library[27] and the XBLOX FPGA library [33]. We summarize the results here; further details can be found in [16].

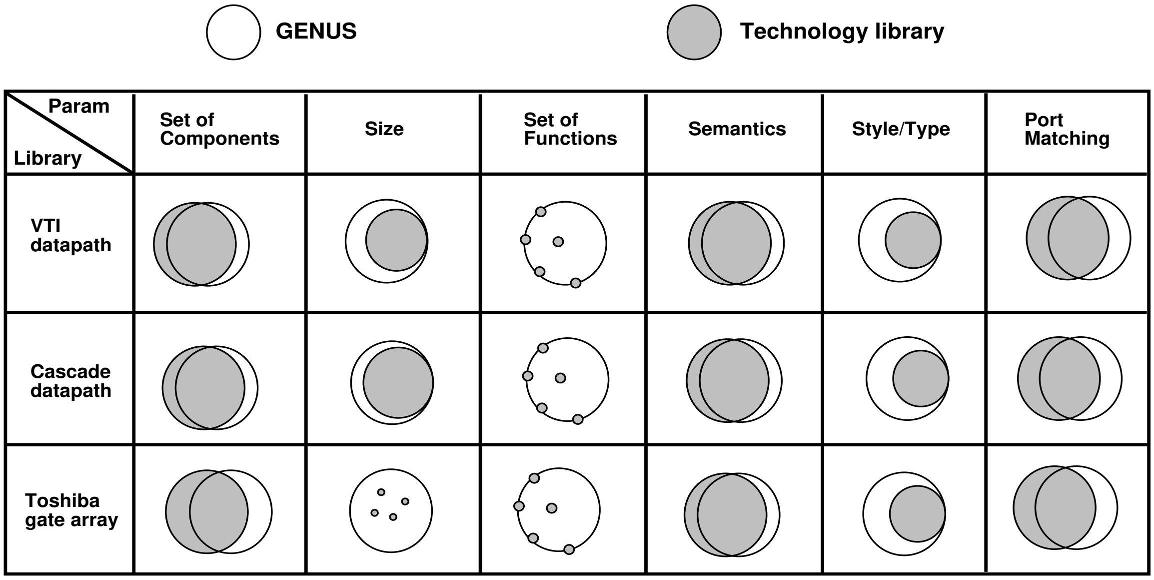

Figure 3: Coverage of GENUS components across different libraries and parameters

Figure 3 pictorially illustrates the coverage of GENUS components across various parameters relative to different technology libraries. [9] contains a description of the set of parameters associated with various components. Each column in Figure 3 represents a parameter type or component attribute and each row shows a technology library. The technology libraries that are parametrized (e.g., Cascade and VTI) provide fairly good coverage for GENUS components. The Toshiba gate array library provides components of specific sizes; components of other sizes have to be built from the available components. The other major difference was the availability of a specific set of functions in realm of multifunction components. We also observed a common problem with mismatches in the port names. This survey indicates that GENUS provides good coverage for several technology libraries.

4.2 Effectiveness of GENUS

In order to further evaluate the usefulness of generic components, we performed some experiments with various designs derived from the HLSW92 benchmark suite [8]. The goal was to test our approach on designs of various sizes, and that encompass different sets of components so as to exercise the major component types in the GENUS component set. In our preliminary set of experiments, the designs varied in complexity from a few hundred gates to a couple of thousand gates. The mapping experiments were performed with respect to two different technology libraries: the VTI Datapath Compiler [29] and the Toshiba Gate Array library [27]. We chose these libraries since they had published gate counts for their databook components, thereby allowing us to compare the effectiveness of mappings for different designs.

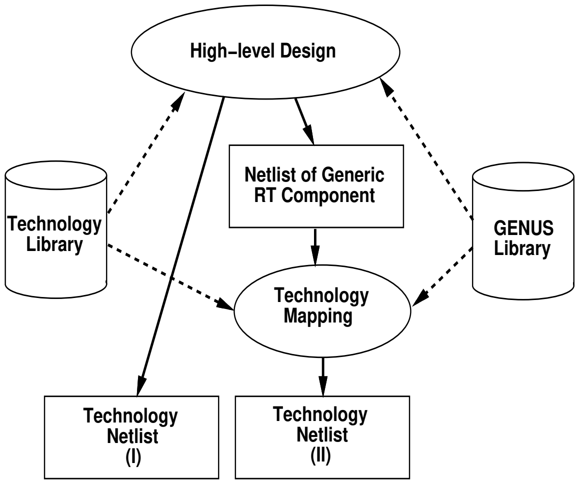

Figure 4: Study of overhead incurred with GENUS

We considered the following RT-level designs derived from different categories such as processors, DSP and interface circuits: the AM2901 Microprocessor[1], the AM2910 Microprogram Controller[2] an SRT Interface[23], and a Circular Buffer(CB) Interface[23].

In our experiments, we designed each of these circuits using three different paths, as shown in Figure 4. First, we designed the circuit using technology library components only (labeled *(I)*). Next, we designed the circuit using only generic components from the GENUS library, and then mapped each of these generic components to the technology library components (labeled *(II)*). The goal was to examine the penalty incurred by designing with a generic component library, followed by technology mapping, as opposed to directly implementing the designs with technology-specific components. For each of these designs, we calculated the total gate count. For our experimental results, a gate is equivalent to the layout area of 4-transistors.

Figure 5 tabulates the gate-counts for various designs across different libraries. For each design and each technology library, we present the gate-count for the two methodologies (I, II). We also present the percentage difference (in terms of the gate-count) between implementing a component in the technology library (I) and mapping the GENUS component design to the technology library (II). This percentage gives a measure of how much overhead is incurred by using generic components from GENUS.

In Figure 5, we observe that the percentage difference between the two design methodologies (I and II) varies from 0.00% to 12.53%. For the SRT interface, the two designs (I and II) are very close in gate-count. This is because the SRT circuit is fairly simple and primarily uses lower-level components such as logic gates and flip-flops that have good coverage relative to the technology libraries. The lack of complex RT components enables a very simple and effective mapping with a resulting overhead that is very low.

Design	VTI Library			Toshiba Gate Array		
	Direct (I)	Tech Map(II)	% Overhead	Direct (I)	Tech Map(II)	% Overhead
AM2901	391	435	11.25%	1523	1628	6.89%
AM2910	832	869	4.44%	1528	1557	1.89%
SRT Interface	735	747	1.63%	674	674	0.00%
CB Interface	1832	1836	0.22%	1979	2227	12.53%

Figure 5: Experimental results of GENUS overhead study

On the other hand, consider the mapping of the 2901 microprocessor and the CB interface to the Toshiba gate array library. There is a significant difference in gate-counts for the two methodologies (I and II). These designs use higher-level components such as ALUs and register-files which have a larger mismatch with respect to the technology library components.

Based on these preliminary experiments, we observe that not much penalty (generally $\leq 10\%$) is incurred in using generic components first and then performing high level library mapping. This supports our hypothesis that high-level mapping is feasible and practical for the designs we examined.

5 Model Generation

The GENUS environment currently provides both automatic simulation model generation and automatic synthesis model generation for each component. The simulation models correspond to behavioral VHDL processes representing functional behavior of the parameterized RT component. The user can also ask for a block-delay model that generates a behavioral VHDL simulation model with a nominal delay for each instantiated component. Since the RT components are generic and are typically used in HLS tasks such as scheduling and allocation, this delay model is often sufficient to support both the design task, as well as the ensuing task of functional verification. The synthesis models for each RT component correspond to Boolean logic equations in a standard form (EQN) for combinational components, and sequentialized EQN for sequential components. These Boolean equations can be used to synthesize generic components from logic-level primitives.

The VHDL simulation models are generated in the order of a few seconds; Figure 6 shows that, even for small examples, the number of VHDL code lines runs into the thousands. Automatic simulation model generation thus obviates the burdensome task of VHDL code generation and

Design	GENUS components	Lines of VHDL code
AM2901	ALU, Reg–file, Reg, Mux	1078
AM2910	ALU, Reg–file, Reg, Mux	854
CB Int	ALU, Reg, Div, Mux	1311
Kalman	ALU, Reg–file, Reg, Mux	1341
Clk–div	ALU, Reg, Mux	476
Timer	ALU, Reg, Mux	605

Figure 6: Behavioral VHDL simulation models

validation, since these model generators have already been validated and tested. The same situation holds for automatic synthesis model generation.

6 Technology Projection

We perform technology projection for different libraries using closed form functions to estimate the area and delay for a component in the generic library. These area and delay models (FA and FD respectively) are functions (a weighted sum) of the parameters(PG) such as bit-width, set-of-functions, etc. required to instantiate a specific component belonging to the generator:

- $FA = a_0 + \sum_k (a_k \star fA_k(PG_k))$
- $FD = d_0 + \sum_k (d_k \star fD_k(PG_k))$

Note that a_0 and d_0 are the constants and a_k and d_k are the coefficients for various terms in these area and delay models.

Our estimation technique uses a sample space of design points on which we perform a least-square regression fit of the formulated functions FA and FD. We believe that this is a useful approach since designers often store the attributes of commonly occurring designs (e.g., 4-, 8- and 16-bit adders). Since a regression analysis using least-square approximation may not capture the intricacies of certain component implementations, we have to pay attention to the appropriate selection of the sample data points. The following steps summarize our approach:

Step 1 *Generate some real design structures for a component and obtain sample data points for area and delay.*

Step 2 *Study the structure of the design generated, and the variations of the area-delay metrics with respect to the parameters that define the component.*

Step 3 *Formulate functions for estimating the area and delay of the generator.*

Step 4 *Run least square approximation to calculate the constants (a_0 and d_0) and the coefficients of various terms (a_i and d_i) used for the functions modeling the metrics.*

Step 5 *Test the estimation model.*

Figure 7 illustrates above steps pictorially along with an example. The area/delay models for the Encoder generator in this figure are functions of generator parameter num-of-inputs (*ni*).

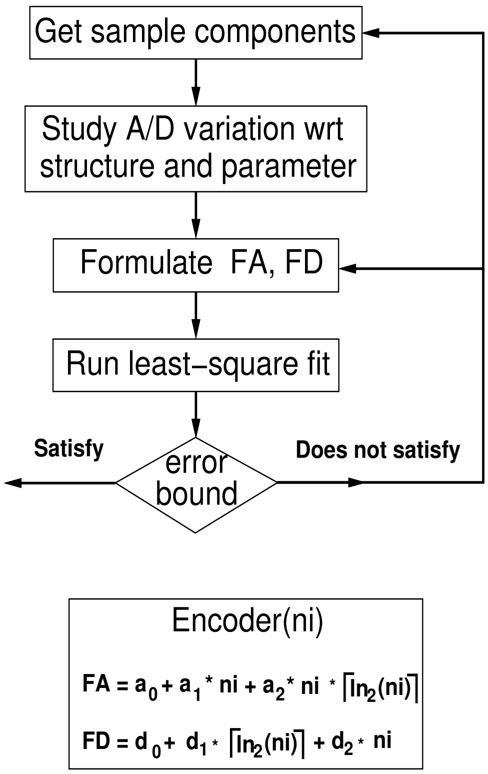

Figure 7: Estimation approach with an example

We test the accuracy of the results against a user-specified error bound. If our models do not satisfy the user-specified error bounds, we go through an iterative experimentation phase, where we repeat some of the Steps 1-5 above. We note that the error bound is often satisfied by the simple addition of linear and logarithmic factors to the estimation functions, as suggested by the design's structure. In an extreme case when the error bound is not satisfied, we may have to generate more implementation data points and repeat all of the steps to obtain new coefficients. However, in our experiments, we have observed convergence within one or two iterations for an error bound of 10%. Further details on the formulation of the models can be found in [15].

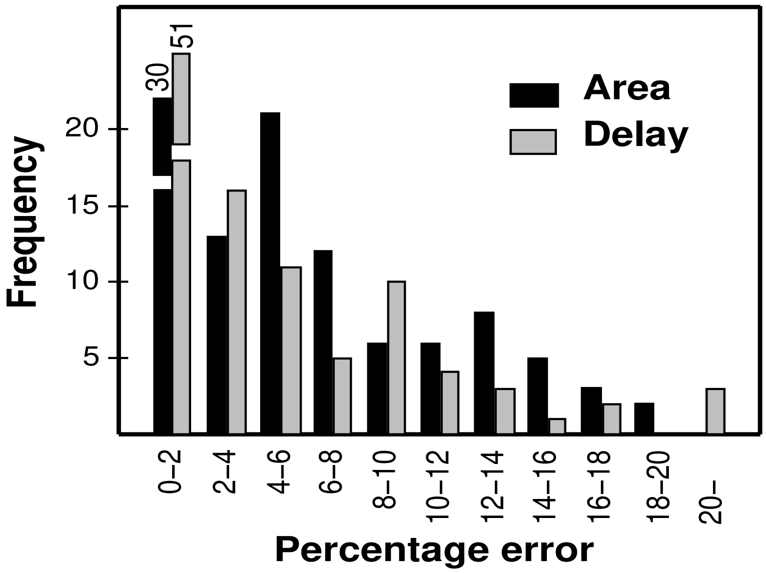

Figure 8: Aggregate error profile as compared to DTAS

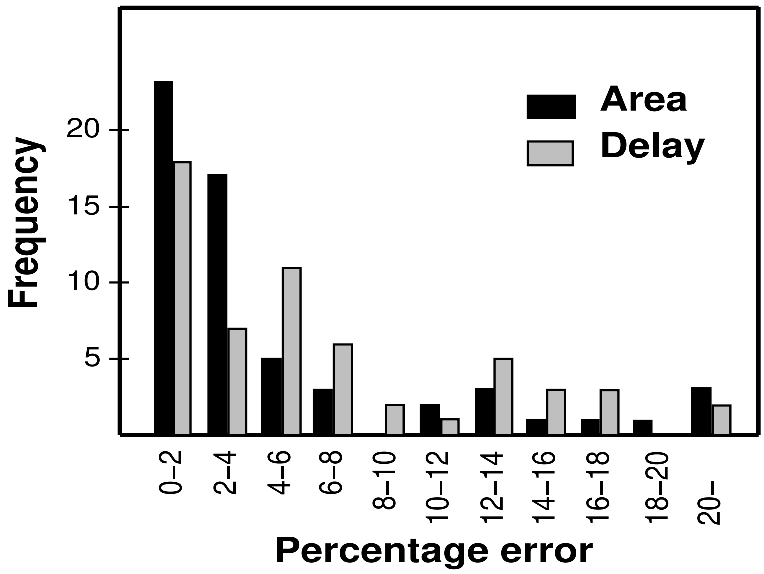

Figure 9: Aggregate error profile as compared to LAST/TELE

We now describe the experiments performed to test our models. We compared the estimates generated by our model against the metrics derived from the design structures generated by DTAS [20], LAST [21] and TELE [25]. The area values provided by DTAS counts the number of equivalent two-input NAND gates used to implement the component. For a component's delay (measured in nanoseconds), DTAS returns the worst-case delay for all paths through the design. LAST and TELE provide area and delay values respectively, based on GDT 3μ CMOS standard cell technology.

Our experiments attempted to cover a wide range of possible component implementations. We did this by generating parameter values randomly for each component generator. The number and set of functions (for multi-function components) were also chosen randomly. For each such randomly chosen component, we ran our models, and compared the results with an actual design generated by above mentioned tools. We considered the following generators: Logic gates, Multiplexer, Comparator, LU, Adder, ALU and Shift-register.

Figures 8 and 9 show the aggregate percentage error profiles across all the generators in consideration; the detailed results for each generators can be found in [14]. From Figure 8 we observe that with respect to DTAS roughly one-third of the area and roughly half the delay data points exhibit an error of less than two percent. After this huge concentration in the range of 0-2%, the frequency of error tapers off as the error increases. For area, *77% percent of the test points have error less than 10 percent* and *95% test points have errors less than 16 percent.* For delay, figures are 87% and 94% respectively. Figure 9 exhibits a similar trend. These results validate our hypothesis about the goodness of our estimators.

We conclude our analysis with two important observations. First, our test points were generated randomly (i.e., we randomly selected the component generator parameter values). Note that in a real design situation, components with certain design properties (i.e., parameters) will be invoked more often, and can be stored with precomputed metrics in the component database. This will lower our average error. Second, our estimation models are integrated on-line with HLS tools. This is possible because of the simple estimation functions chosen, which use only a few additions, multiplications and logarithmic operations. We thus tradeoff accuracy of the metrics (i.e., $\pm 10\%$) for real-time evaluation of the estimates.

7 Summary

With increasing interest in tightly coupling high-level design techniques with physical design, the need for a well-characterized RT-library, together with technology projection using accurate estimation models has emerged. In this work, we motivated the need for such generic RT component libraries, and described GENUS, a specific well-characterized generic RT component library. We then surveyed the relative coverage of GENUS with respect to some technology libraries and performed experiments on some high-level synthesis benchmarks for testing the usefulness of GENUS. In particular, we studied the penalty incurred by using the GENUS generic RT components followed by technology mapping, versus directly implementing the designs in the technology libraries. Our preliminary results are encouraging, indeed even promising, since the maximum overhead we observed was in the range of 10% for area on the benchmarks we examined.

We also presented an automatic model generation technique for simulation and synthesis, as well as a technology projection scheme to link physical design-level information using accurate on-line estimators for the area and delay of the GENUS RT component generators. The simulation and synthesis model generators increase designer productivity, since the models are generated automatically in the order of seconds via component parameters. Furthermore, the model generators reduce design errors as compared to the tedious process of manual model generation, since the generators are pretested and encapsulated. Our estimation models can handle the area/delay contributed by functional blocks as well as the total area/delay including the wiring. We have demonstrated the estimation technique on both combinational and sequential RT components with aggregate errors in the range of $\pm 10\%$. Our model generators are simple, fast and fairly accurate, and have been integrated with an existing high-level synthesis system [26].

We believe that the benefits of using a standard component set such as GENUS greatly outweigh the small penalty that may be incurred during technology mapping to target libraries. Coupled with technology projection, this approach can effectively support accurate system-level design space exploration. We demonstrated a concrete technique for linking system-level design information with different technology libraries using the GENUS RT component library. Future work needs to address the high-level technology mapping problem and the tradeoffs between custom synthesis versus standard component realization.

8 Acknowledgements

This work was supported in part by in part by NSF grant #MIP9009239 and in part by SRC contract #93-DJ-146. We also thank Prof. Daniel Gajski for his helpful comments.

References

[1] "Am2901c: Four-bit Bipolar Microprocessor Slice," *Applied Micro Devices Inc., California,* 1993.

[2] "Am2910A: Microprogram Controller," *Applied Micro Devices Inc., California,* 1993.

[3] "Cascade Design Automation Databook," *Cascade Design Automation, Washington,* 1992.

[4] R. Camposano and L. H. Trevillyan, "The Integration of Logic Synthesis and High-Level Synthesis," *Proc. ISCAS,* 1989.

[5] V. Chaiyakul, A. C-H. Wu and D. D. Gajski, "Timing Models for High-level Synthesis," *Proc. of The European Design Automation Conference,* September 1992.

[6] H. De Man, et. al., "Architecture-Driven Synthesis Techniques for VLSI Implementation of DSP Algorithms," *Proceedings of the IEEE,* Vol. 78, No. 2, pp. 319-335, February 1990.

[7] N. D. Dutt and J. R. Kipps, "Bridging High-Level Synthesis to RTL Technology Libraries," *Proc. 28th Design Automation Conference,* June 1991.

[8] N. D. Dutt and C. Ramachandran, "Benchmarks for the 1992 High Level Synthesis Workshop," *Technical Report 92-107, University of California at Irvine,* 1992.

[9] N. D. Dutt, "GENUS:A Generic Component Library for High Level Synthesis," *Technical Report 88-92, University of California at Irvine,* 1988.

[10] "EDIF 2 0, Library of Parametrized Modules," *Electronic Industries Association, Washington, D.C.* 1993.

[11] D. Gajski, N. Dutt, A. Wu and S. Lin, "High-Level Synthesis: Introduction to Chip and System Design," *Kluwer Academic Publishers* 1992.

[12] The IEEE, "IEEE Standard VHDL Language Reference Manual," IEEE, 1987.

[13] R. Jain, "MOSP: Module Selection for Pipelined Designs with Multi-cycled Operations," *Proc. IEEE International Conference on Computer-aided Design,* pp. 212-215, November 1990.

[14] P. K. Jha and N. D. Dutt, "A Fast Area-Delay Estimation technique for RTL component generators," *Technical Report 92-33, University of California at Irvine,* April 1992.

[15] P. K. Jha and N. D. Dutt, "Rapid Estimation for Parametrized Components," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* pp. 296-303, September 1993.

[16] P. K. Jha, N. D. Dutt and D. D. Gajski, "An Evaluative Study of RT Component Libraries," *Technical Report 93-11, University of California at Irvine,* March 1993.

[17] P. K. Jha, T. Hadley and N. D. Dutt, "The GENUS User Manual and C Programming Library," *Technical Report 93-32, University of California at Irvine,* April 1993.

[18] Q. Ji, Y. S. Oh, M. R. Lightner and F. Somenzi, "Technology Independent Estimation of Area in Logic Synthesis," *Proc. SASIMI,* April 1992.

[19] S. Kang and S. Szygenda "Automatic VHDL Model Generation," *Proc. IFIP Conference on Hardware Description Languages and their Applications,* April 1993.

[20] J. R. Kipps, "An Approach to Component Generation and Technology Adaption," *Ph.D. Dissertation, University of California at Irvine,* December 1991.

[21] F. J. Kurdahi and C. Ramachandran, "LAST: A Layout Area and Shape function esTimator for High Level Applications," *Proc. European Design Automation Conference*, February 1991.

[22] G.W. Leive and D.E. Thomas, "A Technology Relative Logic Synthesis and Module Selection System," *Proc. 18th Design Automation Conference,* 1981.

[23] J. Li, "VHDL Modeling for Silicon Compilation," *M.S. Thesis, University of California at Irvine,* 1993.

[24] P. Michel, U. Lauther and P. Duzy (Editors) "The Synthesis Approach to Digital System Design," *Kluwer Academic Publishers,* 1992.

[25] C. Ramachandran and F. Kurdahi, "TELE: A Timing Evaluator using Layout Estimation for High Level Applications," *Proc. European Design Automation Conference*, March 1992.

[26] L. Ramachandran and D. D. Gajski, "An Algorithm for Component Selection in Performance Optimized Scheduling," *Proc. International Conference on Computer-aided Design*, November 1991.

[27] "Toshiba ASIC Gate Array Library," *Toshiba Corporation, Tokyo, Japan,* 1990.

[28] A. Tyagi, "A Module Generator Development Environment: Area Estimation and Design-Space Exploration Encapsulation" *Proc. 6th International Conference on VLSI Design, Bombay,* January 1993.

[29] "VDP300 CMOS Datapath Library," *VLSI Technology, Inc., San Jose, California,* November 1991.

[30] D. E. Wallace and M. S. Chandrasekhar, "High-level Delay Estimation for Technology-Independent Logic Equations," *Proc. International Conference on Computer-aided Design*, November 1990.

[31] A. C-H. Wu, V. Chaiyakul and D. D. Gajski, "Layout-Area Models for High-Level Synthesis," *Proc. International Conference on Computer-aided Design*, November 1991.

[32] Wayne H. Wolf, "How to Build a Hardware Description and Measurement System on an Object-Oriented Programming Language," *IEEE Transaction on Computer-aided Design*, pp. 288-301, March 1989.

[33] S. H. Kelem and J. P. Seidel, "Shortening the Design Cycle for Programmable Logic Devices," *IEEE Design and Test of Computers,* pp. 40-50, December 1992.