BRIDGING HIGH-LEVEL SYNTHESIS TO RTL TECHNOLOGY LIBRARIES

Nikil D. Dutt and James R. Kipps

Dept. of Information & Computer Science
University of California, Irvine
Irvine, CA 92717

Abstract

The output of high-level synthesis typically consists of a netlist of generic RTL components and a state sequencing table. While module generators and logic synthesis tools can be used to map RTL components into standard cells or layout geometries, they cannot provide technology mapping into the data book libraries of functional RTL cells used commonly throughout the industrial design community. In this paper, we introduce an approach to implementing generic RTL components with technology-specific RTL library cells. This approach addresses the criticism of designers who feel that high-level synthesis tools should be used in conjunction with existing RTL data books. We describe how GENUS, a library of generic RTL components, is organized for use in high-level synthesis and how DTAS, a functional synthesis system, is used to map GENUS components into RTL library cells.

1. Introduction

High-level synthesis systems transform an abstract behavioral specification into a structure of RTL operators and a state sequencing table. Most high-level synthesis systems map operators to generic RTL components to effect technology independence. The output of high-level synthesis is then input to logic- and layout-level synthesis tools to complete the design. Although abstract component characterization is an important task in high-level synthesis, most existing high-level synthesis systems have not explicitly addressed this issue. Traditional high-level synthesis systems either use very abstract components, which provides crude estimates for delay and area, or use design components from a particular technology library, which yields good performance estimates but complicates the task of retargetting to new libraries.

In this paper, we outline a novel method for coupling *technology independence in high-level synthesis* with *technology mapping to RTL library cells*. Technology independence is achieved through the use of **GENUS** and **LEGEND**, while technology mapping to RTL library cells is achieved through the use of **DTAS**. GENUS is a parameterizable library in which generic components are instantiated by specifying parameters that define their structural, operational, and performance attributes. LEGEND is language that allows the specification new GENUS libraries, as well as the customization of existing libraries. DTAS is a rule-based system for functional synthesis of generic RTL components, such as those found in a GENUS library.

The combination of GENUS, LEGEND, and DTAS allow us to address the criticism that high-level synthesis tools are not useful because they do not map designs into the RTL components available in technology-specific data books. Such components are used in many existing industrial designs for implementing regular structured logic, such as ALU's, multipliers, and counters. For this design scenario, current high-level synthesis techniques are not immediately useful, since there is no automatic path available to map generic components into technology-specific RTL components. In this paper, we show a path for implementing generic components from high-level synthesis by treating data book components as RTL library cells.

2. Previous Work

Some work on characterization of technology-specific components and module databases has been done at the layout and logic levels [LeTh81] [Wolf86]. However, not much attention has been focused on characterizing generic components at the register-transfer level. VHDL [IEEE87] is a proposed standard for design documentation and exchange. Although VHDL has good constructs for describing specific libraries and component instances, it is unsuitable for fully describing customized, parameterizable component libraries.

Silicon compilers and behavioral synthesis tools typically use the module generator approach to RTL component synthesis, where module generators are integrated with logic synthesis tools [CaTr89]. For instance, the Yorktown silicon compiler [BrCa88] maps combinational operators to modules; these modules are then input to logic synthesis tools using ESPRESSO-II [BrHM84]. Similarly, ICDB [ChGa90] uses component generators to produce logic equations for each functional component; these equations, together with performance constraints, are input to the MILO logic optimizer [VaGa88]. These systems take a "procedural" view of RTL synthesis that does not preserve the hierarchical structure of component design for

technology mapping. Hence, technology mapping is done at the logic level on large flat designs, which requires DAG matching by detecting isomorphism of large subgraphs [Keut87]. This complicates the task of interfacing to a given cell library that may consist of large cells at the MSI and LSI level.

3. System Overview

Figure 1 shows the overall framework of a system that performs high-level synthesis and that implements generic components using RTL library cells. The design is initially specified in an abstract behavioral language. The generic component library, GENUS, is generated from a LEGEND description. High-level synthesis tools such as state schedulers, component allocators, component and connectivity binders, progressively transform the abstract behavioral design specification into a state sequencing table and a netlist of GENUS components described using structural VHDL.

The state sequencing table is accepted by a control compiler that extracts the sequencing logic and applies logic-level optimizations and technology mapping techniques. The GENUS netlist, consisting of regular-structured data path components such as MUXs, ALUs, and registers, is input to DTAS. This netlist is translated into an internal representation that is then passed through a phase of functional decomposition and technology mapping.

The output of DTAS is a set of alternative implementations of the input netlist. Each implementation is represented as a hierarchical netlist that traces the top-down design of the input netlist into subcomponents. Leaves of each hierarchical netlist map the alternative

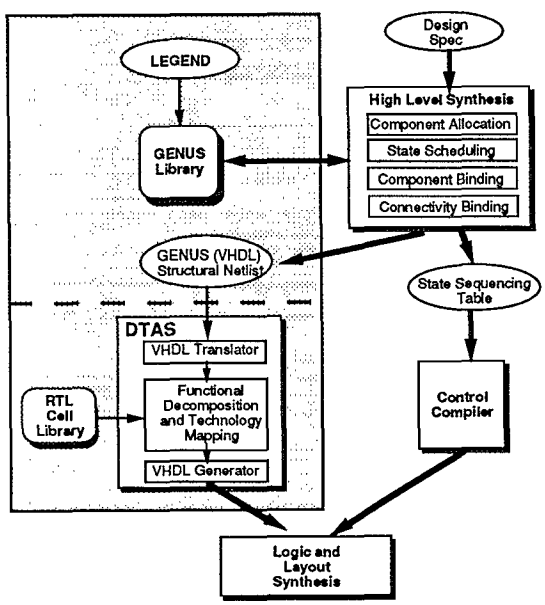

Figure 1. System Architecture

design to cells drawn from the given RTL library. Netlists vary by the design styles and library cells used in their construction. The hierarchical netlists can be output in structural VHDL and passed to other tools for analysis, optimization, and layout.

4. GENUS and LEGEND

GENUS [Dutt88] is a framework for maintaining and accessing libraries of generic RTL components. LEGEND [Dutt90] is a generator-specification language for describing the contents of a GENUS library. Each generic component generator is characterized by a unique name and a list of parameterizable attributes. Typical parameters include the component's *style*, *functionality*, *input-output characteristics*, *size*, *bit-width*, and *representation*.

The LEGEND description can be tailored to a particular generic component library by specifying the necessary component generator types. In addition, each component generator can produce simulatable VHDL behavioral models for the generated components. These models can be used to verify the behavior of a synthesized design. Figure 2 shows a typical LEGEND description for a generic counter component.

A GENUS library is composed as a hierarchy of types, generators, components and instances. The *type* class describes the abstract functionality of elements in GENUS. Sample type attributes include *combinatorial*, *sequential*, *interface*, and *miscellaneous*.

```
NAME:                    COUNTER
CLASS:                   Clocked
MAX_PARAMS:              7
PARAMETERS: GC_COMPILER_NAME, GC_INPUT_WIDTH (%w),
            GC_NUM_FUNCTIONS, GC_FUNCTION_LIST,
            GC_SET_VALUE, GC_STYLE, GC_ENABLE_FLAG
NUM_STYLES:              2
STYLES:          SYNCHRONOUS, RIPPLE
NUM_INPUTS:              1
INPUTS:          I0[%w]
NUM_OUTPUTS:     1
OUTPUTS:                 O0[%w]
CLOCK:                   CLK
NUM_ENABLE:              1
ENABLE:          CEN
NUM_CONTROL:     3
CONTROL:                 CLOAD, CUP, CDOWN
NUM_ASYNC:               2
ASYNC:                   ASET, ARESET
NUM_OPERATIONS: 3
OPERATIONS:
        (        (LOAD)
                 (INPUTS: I0)
                 (OUTPUTS:       O0)
                 (CONTROL:       CLOAD)
                 (OPS:   (LOAD:  O0 = I0)))
        (        (COUNT_UP)
                 (OUTPUTS:       O0)
                 (CONTROL:       CUP)
                 (OPS:   (COUNT_UP: O0 = O0 + 1)))
        (        (COUNT_DOWN)
                 (OUTPUTS:       O0)
                 (CONTROL:       CDOWN)
                 (OPS: (COUNT_DOWN: O0 = O0 - 1)))
VHDL_MODEL:              counter_vhdl.c
OP_CLASSES:              default
```

Figure 2. LEGEND Counter Generator Description

A *generator* class is used to generate a family of similar components and instances. LEGEND descriptions are used to maintain lists of all possible parameters and definitions for every possible operation performed by a generated component. A *component* is generated by passing a list of parameters to the parent LEGEND generator descriptor; some parameters are obligatory, others may be assigned a default value. *Instances* are "carbon-copies" of a generated component, with unique names. GENUS component instances are used in the final structural design produced by high-level synthesis tools. Since an instance inherits all of its attributes from the parent component, only the connectivity of the instance is stored in its representation. Table 1 shows some typical components available in the GENUS/LEGEND environment.

5. DTAS

GENUS component instances are mapped into technology-specific designs by DTAS [KiGa91]. The input to DTAS is a *netlist* of instantiated GENUS components (or *modules*), which is passed through a phase of functional decomposition and technology mapping. Functional decomposition is implemented with a rule-based system that expands the space of component decompositions. This design space is represented as an acyclic graph. Nodes consist of component specifications and alternative component implementations. Each component implementation corresponds to a library cell or to a netlist of modules. A netlist represents one level of component decomposition; its modules represent connected subcomponents. Each module is described by a component specification and will be mapped to one implementation of that specification. The output of DTAS is a set of hierarchical, library-specific netlists that represent alternative implementations of the components in the input netlist.

Combinational		Sequential
Boolean Gates	Comparator	Register
LU	ALU	Register File
Mux	Shifter	Counter
Selector	Barrel Shifter	Stack/FIFO
Decoder	Multiplier	Memory
Encoder	Divider	
Adder/Subtractor		
Interface		**Miscellaneous**
Port		Bus
Buffer		Delay
Clock Driver		Switchbox Concat
Schmidt Trigger		Switchbox Extract
Tristate		Clock Generator
		Wired-or

Table 1. Typical LEGEND/GENUS Generic Components

Technology mapping is performed using the *functional* specification of library cells, as opposed to a DAG description of their Boolean behavior. The functionality of library cells, i.e., their type, bit-width, and other characteristics, is described with the same representation language used in recognizing and decomposing GENUS components. During decomposition, component specifications are compared to the functional specification of available library cell; matching cells are mapped into the design space. For example, after DTAS decomposes a 16-bit adder into four 4-bit adders, it examines the cell library for a cell of type ADD with two 4-bit inputs plus carry-in and a 4-bit output plus carry-out. If such a cell exists, it is mapped into the design as an alternative implementation of the 4-bit adder. By performing a functional match, we avoid the complexity of subgraph isomorphism inherent in DAG matching.

If unconstrained, the size of the design space for a given input netlist is the product of the number of alternative implementations for each module in the netlist. Even for components of modest size, such as a 16-bit adder, there can be several hundred thousand to several million alternative designs, only a small percentage of which are of any real interest. Thus, some form of search control is required to limit the size of the design space. We constrain design space expansion in two ways. First, we ignore netlist implementations containing two or more modules with the same component specification that are not instances of the same component implementation. Second, we apply *performance filters* to eliminate all but the "best" alternative implementations of each component specification in the design hierarchy. These two search control principles reduce the design space to a manageable size. For instance, the design space of a 16-bit adder is reduced to ten alternatives designs.

6. Performance Results

Figure 3 shows a comparison of five alternative designs for a 64-bit, 16-function ALU. These designs were generated by DTAS using a subset of 30 cells from LSI Logic Inc.'s macrocell data book [LSIL87]. This set includes 2-to-1, 4-to-2, and 8-to-4 multiplexers, 1-, 2-, and 4-bit adders plus 4-bit carry look-ahead generators, a 2-bit adder/subtractor, D flip flops, and 4- and 8-bit data registers.

The performance filter used in this example accepts all design alternatives that make favorable tradeoffs between area (in equivalent NAND gates) and delay (in nanoseconds). The fastest design alternative is 34 percent larger than the smallest but reduces delay by 81 percent. More significantly, DTAS finds two other alternative designs that reduce delay nearly as well as the fastest but suffer only a 14 percent increase in area. DTAS generated this design space in less than 15 minutes of real time on a SUN-3 workstation.

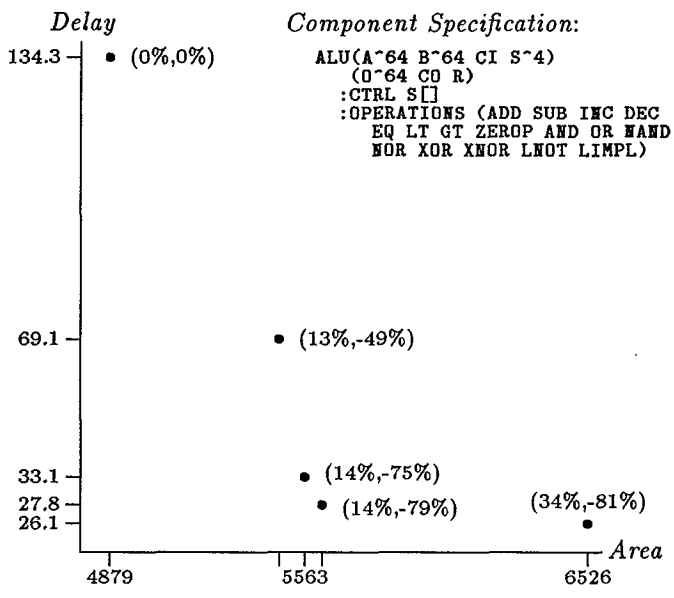

Figure 3. Alternative Designs for 64 bit ALU

7. Status and Future Direction

The GENUS/LEGEND generic component library environment is implemented in C/UNIX and Lex/Yacc on SUN workstation environment. High-level synthesis tools such as VSS [LiGa88] currently use generic components from this environment. The output of the high-level synthesis tools is a VHDL structural netlist of GENUS components, and a state table in control-based BIF [DuHG90] that controls these GENUS components and that sequences the design.

DTAS is implemented in Common Lisp and Rack [Kipp90] and is capable of synthesizing a wide range of RTL components, including bitwise logic gates and multiplexers, binary and BCD decoders and encoders, n-bit adders and comparators, n-bit arithmetic logic units, shifters, n-by-m multipliers, and up/down counters. These components are supported by 86 rules written in the DTAS Design Language. DTAS requires nine library-specific design rules to fully utilize the subset of cells from LSI Logic mentioned above.

To ease the task of moving DTAS into new cell libraries, we are developing **LOLA (Logic Learning Assistant)** [KiGa90]. The purpose of LOLA is to partially automate the maintenance of DTAS's library-specific rules. LOLA is invoked when DTAS is presented with a new cell library or as technology upgrades cause changes in a familiar library. LOLA applies abstract design principles to generate library-specific rules. LOLA then uses these generated rules to modify DTAS's rule base so that DTAS can take advantage of the library changes.

8. Acknowledgements

This research was supported in part by NSF RIA MIP-9009239 and SRC Contract 90-DJ-146. The authors would especially like to thank Prof. Daniel D. Gajski for his insightful discussions and guidance.

9. References

[BrHM84] R.K. Brayton, et al., "ESPRESSO-IIC: Logic Minimization Algorithms for VLSI Synthesis," *Kluwer Academic Publishers*, Netherlands, 1984.

[BrCa88] R.K. Brayton, et al., "The Yorktown Silicon Compiler System," *Silicon Compilation*, D. Gajski (ed.), Addison-Wesley, 1988.

[CaTr89] R. Camposano, and L.H. Trevillyan, "The Integration of Logic Synthesis and High-Level Synthesis," *ISCAS* 89.

[ChGa90] G.D. Chen and D.D. Gajski, "An Intelligent Component Database for Behavioral Synthesis," *27th DAC*, Orlando, July 1990.

[Dutt88] N.D. Dutt, "GENUS: A Generic Component Library for High Level Synthesis," *TR* 88-22, U.C. Irvine, Sept. 1988.

[Dutt90] N.D. Dutt, "LEGEND: A Language for Generic Component Description," 1990 *IEEE International Conference on Computer Languages*, New Orleans, March 1990.

[DuHG90] N.D. Dutt, T. Hadley and D.D. Gajski, "An Intermediate Representation for Behavioral Synthesis," *27th DAC*, 1990.

[IEEE87] *IEEE Standard VHDL Language Reference Manual*, IEEE, 1987.

[Kipp90] J.R. Kipps, "RACK: A Parser Generator for AI Languages," *IEEE International Conference on Tools for AI*, Washington DC, Nov. 1990.

[KiGa90] J.R. Kipps and D.D. Gajski, "The Role of Learning in Logic Synthesis," *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 4, No. 2, World Scientific Press, June 1990.

[KiGa91] J.R. Kipps and D.D. Gajski, "Automating Technology Adaptation in Design Synthesis," *Applications of Learning and Planning Methods*, N.G. Bourbakis (ed.), World Scientific Press, 1991.

[Keut87] K. Keutzer, "DAGON: Technology Binding and Local Optimization by DAG Matching," *24th DAC*, 1987.

[LeTh81] G.W. Leive and D.E. Thomas, "A Technology Relative Logic Synthesis and Module Selection System," *18th DAC*, 1981.

[LSIL87] LSI Logic, Inc., "Databook: 1.5-Micron Compacted Array Technology," 1987.

[LiGa88] J.S. Lis and D.D. Gajski, "Synthesis From VHDL," *ICCD* 88, *Oct.* 1988.

[VaGa88] N. Vander Zanden and D.D. Gajski, "MILO: A Microarchitecture and Logic Optimizer," *25th DAC*, 1988.

[Wolf86] W. Wolf, "An Object-Oriented Procedural Database for VLSI Chip Planning," *23rd DAC*, 1986.