

# METHOD FOR AUTOMATIC GENERATION OF BRANCHING PROGRAMS USING DECISION DIAGRAMS

*Stanislav Stanković<sup>1</sup> and Jaakko Astola<sup>2</sup>*

<sup>1</sup>Tampere International Center for Signal Processing, Tampere University of Technology,  
P.O.Box 553, FIN-33101 Tampere, FINLAND, stanislav.stankovic@tut.fi

<sup>2</sup> Institute of Signal Processing, Tampere University of Technology,  
P.O.Box 553, FIN-33101 Tampere, FINLAND, jaakko.astola@tut.fi

## ABSTRACT

Distinction between behavioral and structural models is one of the key points in high-level synthesis and modern hardware design.

Decision Diagrams are known to be an efficient way of representation of switching functions. In recent years they have become very important tool in fields of logic designs for the tasks of logic circuit design, testing and verification. Correspondence between decision diagrams and structural hardware models is well known. On other hand decision diagrams can be easily converted into high level formal language branching programs, which can be viewed as behavioral system models. The ability to freely flow between these two modeling strategies illustrates the versatility of decision diagrams as a form of representation.

In this paper, we explore the relationship of recently introduced [30], [29] XML based system for representation of Decision Diagrams with notions of behavioral and structural models. We examine this problem on example of automatic synthesis of branching programs.

A large number of systems for manipulation of decision diagrams has been proposed over the years. In our previous work we have proposed an XML based system for representation of various types of decision diagrams. This system was designed with robustness and flexibility in mind, permitting easy conversion of an abstract general decision diagram representation to application specific tasks. We further expand these properties of the systems by introducing an extension that is capable of automatic generation of branching programs in C/C++ structure, based on given discrete functions.

## 1. INTRODUCTION

In recent years, the size and complexity of logic circuits has reached such level of complexity that conventional models of representation and analysis have become impractical. Decision Diagrams have been proposed in, [4], [20], [23], as an efficient, in terms of processing time and needed memory space, method of representation of switching functions.

In this paper, we explore some of the questions opened in [30], namely the position of decision diagrams, as a

form of representation of discrete functions, in relation to behavioral and structural hardware models. We discuss this problem by examining the relations of Shannon binary decision diagrams and branching programs.

In our previous work we have proposed XML:DD, an XML based framework for description of the structure of decision diagrams. The idea behind this proposal was to introduce a uniform way of representation, a sort of a file format, for various types of decision diagrams. The main idea is to design a system robust enough to represent different classes of decision diagrams, and flexible enough to permit easy transformation of this general abstract representation into application specific formats. We have illustrated these properties of the proposed system in [29], on an example of automatic visualization of decision diagrams. In [30] we have further demonstrated the usability of the system on an example of automatic generation of hardware models in VHDL.

In this paper, we propose an extension to the original system capable of converting a decision diagram represented in XML form into a branching program in some high-level programming language such as C/C++. For this task, we use a mechanism, based on XSLT, similar in structure to ones that were used in [29], [30]. We compare the properties of this system to the structural hardware description models in VHDL obtained by the system introduced in [30].

This paper is organized in the following way. We recall some basic notions about decision diagrams in section 2. The relationship between decision diagrams and branching programs is discussed in theoretical terms in this section also. In section 3 we give a brief overview of related work by other authors. Hardware implementation of switching functions using decision diagrams is given in section 4. Finally, section 5 focuses on the XML based system for automatic generation of branching programs. We present an illustrative example of a branching program generated for a given boolean function. Possibilities of generalization of proposed method to other types of decision diagrams are also presented in this section. We give conclusion and closing remarks in section 6.

## 2. DECISION DIAGRAMS, STRUCTURAL VS. BEHAVIORAL REPRESENTATION

In order to clearly understand the question at hand it is necessary to recall some basic notions regarding decision diagrams.

By definition, decision diagrams for representation of discrete functions are derived reducing decision trees by applying a set of properly formulated reduction rules [4], [24], [26]. The rules for reduction of decision trees depend on the type of the decision tree in question [24]. In general, the reduction is performed by identifying identical sub-trees in the decision tree. Only one copy of the sub-tree is kept. Edges connecting nodes to the removed sub-trees are rearranged to point to the remaining copy.

Formally, a decision diagram is an acyclic directed graph consisting of nodes and edges that connect them. In representations, nodes must store information about the incoming and outgoing edges as well as some additional information, depending on the type of the decision diagram. Each decision diagram has one root node representing the beginning of the directed graph, and a set of terminal nodes. From the point of view of data structures, terminal nodes differ from non-terminal ones only in the fact that they do not have any outgoing edges.

Different types of decision diagrams have been defined depending on the number of outgoing edges per node permitted, and used decomposition rules [4], [9], [14], [21], [23], [26], [27].

In this paper we focus on a special class of decision diagrams, so called Shannon binary decomposition diagrams. This particular class of decision diagrams is defined of the class of binary switching functions,  $C_2^n \rightarrow GF(2)$ , by recursive application of Shannon decomposition rule. Consider a switching function  $f(x_1, x_2, \dots, x_n)$ . Shannon decomposition rule for any variable  $x_i$  can be defined as follows:

$$f = \bar{x}_i f_0 \oplus x_i f_1, \quad f_0 = f|_{x_i=0}, \quad f_1 = f|_{x_i=1} \quad (1)$$

It is evident from previous equation that the value of logic variable  $x_i$  determines the value of the output  $f$ , setting it either to  $f_0$  or to  $f_1$ . In essence, the value of variable  $x_1$  regulates the flow of information carried by  $f_0$  and  $f_1$ . A structure that regulates the flow of execution of programming code in structural programming languages in the equivalent manner is a well known If-Then-Else structure. There is a clear and intuitive correspondence between Shannon decomposition and If-Then-Else programming structure. The first term of resulting decomposition, the one obtained for value  $x_i = 0$ , corresponds to the Then branch of the If-Then-Else structure. The Else branch is equivalent to the second term of the decomposition,  $x_i = 1$ . Value of variable  $x_i$  is a condition that determines the order choice of execution path.

By recursive application of this decomposition rule we obtain a full decomposition tree. Decomposition diagram obtained by complete reduction of this tree corresponds to

the optimal branching program that implements the logic function  $f(x_1, x_2, \dots, x_n)$ .

Therefore, we can say that a binary decomposition diagram is a graph-line data structure which describes a behavior of a certain system, not by data stored in its constituent components, since all nodes represent an identical operation, but by its topology. By extension, the same statement holds for any other type of decision diagrams.

The fact that we are able to make a smooth transition between structural and behavioral model, highlights an important duality inherent to decision diagrams, with significant implications on hardware design.

We will demonstrate this property using an XML based mechanism to automatically generate branching programs in C/C++ syntax, based on given decision diagrams. The XML representation of decision diagrams is identical to the one used in [30] to automatically generate RTL models using VHDL syntax. Furthermore, a similar conversion mechanism based on XSLT is used in both cases. This serves as a good demonstration of the proposed XML based representation framework.

The correspondence between decision diagrams and branching programs has been discussed in great detail in [2] and [17].

## 3. RELATED WORK

The use of XML for representation of decision diagrams, as proposed in our previous papers [29], [30], is a form of representation at a very high level of abstraction. One aims at encapsulating a large amount of information in a way suitable for easy conversion to different application specific format. In general this system stands in line with a larger problem high-level hardware synthesis. Fig. 1 presents the diagram of a typical hardware design system.

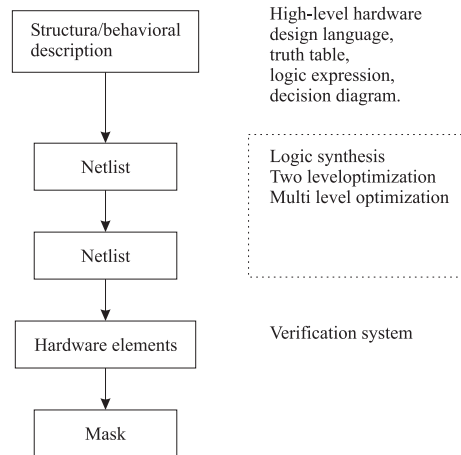


Figure 1. Hardware design system

The problem of translating a high-level language behavioral or structural description into a register transfer level (RTL) representation is called high-level synthesis [10]. The focus in this field has been mainly on developing tools that are able to utilize behavioral descriptions of

various hardware systems created in high-level programming languages such as C/C++ [8], [15].

There have been several commercial efforts to develop compilers taking C/C++ into VHDL or Verilog including CoWare, Adelante [1], Celoxica [6], C Level Design [32], and Cynapps [7]. SystemC developed by SystemC Consortium [22] represents a C/C++ based language designed especially for this purpose. There are tools as for example Concentric by Synopsys, that take in the SystemC code as an input and produce RTL models in VHDL and Verilog. The MATCH compiler developed by P. Banerjee at Northwest University is a behavioral synthesis tool for DSP applications that produces RTL models in VHDL or Verilog based on MATLAB behavioral description [3], [15].

#### 4. HARDWARE IMPLEMENTATION OF DISCRETE FUNCTIONS USING DECISION DIAGRAMS

In the previous section we have established the correspondence between Binary Decision Diagrams and branching programs. We begin this section by stating the correspondence between binary decision diagrams and hardware implementations of switching functions. In Fig. 1 we show a circuit that implements a Shannon decomposition rule. Hardware implementation of a binary function using decision diagrams would represent a hierarchy of such objects. Each node in the decision diagram would be replaced by one such circuit. The internal connections between these elements correspond to the edges of the decision diagram.

The system we proposed in our previous papers is capable of generating hardware architectures for Shannon binary decision diagrams.

Since the basic functionality is repeated for every node in the decision diagram, we can reuse the same basic logic circuits in the hardware implementation. Corresponding hardware architecture is a network of interconnected instances of these components. The actual implementation of basic expansions can be developed using a wide range of technologies from ASIC to LUT based FPGAs. Hardware realization of a logic function,

$$f(x_1, x_2, x_3) = x_1\bar{x}_2 \vee x_2x_3, \quad (2)$$

using the corresponding BDD can be seen in Fig 2.

For the detailed description of VHDL implementation of such circuit please refer to [30]. The hardware architecture organized in this way represents clearly a structural hardware description.

#### 5. XML BASED SOLUTION

XML [13], [12] represents a special purpose markup language. It is intended for the description of formats for recording various kinds of data with a complex internal structure and possibly mixed information content. Rather than trying to specify a rigid data structure for any possible kind of data, and for each particular application, XML

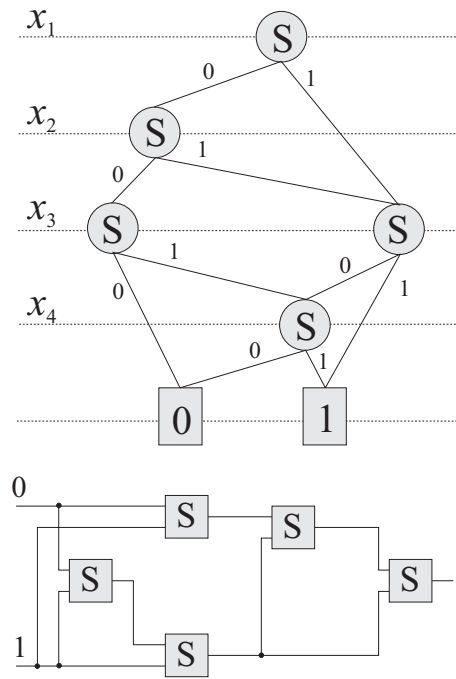


Figure 2. BDD and a logic network corresponding to the function  $f(x_1, x_2, x_3) = x_1\bar{x}_2 \vee x_2x_3$ .

specifies a simple set of rules that an XML based file format should use in order to be understood by the XML enabled software. Each XML document is a structured text, while the information it contains is presented in the form of hierarchically arranged elements. The type of elements and their internal structure is specified separately according to the syntax of XML rules.

This XML description can easily be converted to different application specific formats, via XSLT, special subset of XML standard. The structure of XML:DD framework is presented in Fig. 3. We discuss the main elements of the proposed system in the following sections.

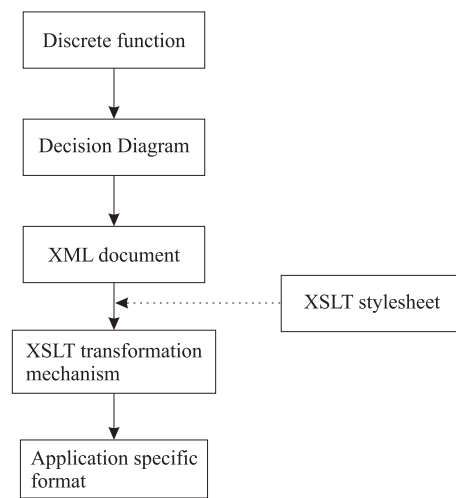


Figure 3. Structure of the XML:DD framework

## 5.1. Basic Notions on XML based framework for Decision Diagrams

The focus in development of this framework was on ability to store as much information about the topology of a decision diagram in as compact as possible form. We will briefly go through the basics of the mutual structure of all the XML:DD documents.

XML documents store data in the form of the structured text. Elements of the text form a hierarchy. In the case of XML:DD the root element of the hierarchy is `<dd:tree>` element. This element serves as a container for the decision diagram itself. It also stores some important information about the nature of the diagram, most importantly the type of the diagram and the number of levels the diagram contains. The number of levels of a decision diagram is equivalent to the number of logical variables of the discrete function this diagram represents.

Nodes of decision diagram are represented in form of `<dd:node>` elements. Each node contains another `<dd:node>` sub element corresponding to the next node in in-order recursive traversal of the decision tree. Recursive structure of a decision diagram is thus mapped directly to the recursive properties of XML. Two separate linked lists representing parents and children nodes are attached to each `<dd:node>` element. The elements representing non-terminal nodes contain the following attributes:

1. ID - unique identifier of each node
2. Level in a diagram to which the node belongs
3. Rule - An indicator of an expansion (decomposition rule) that was applied to this node. Since we are dealing only with Shannon binary decision diagrams, the value of this attribute is constant.

Terminal nodes are stored using the same kind of `<dd:node>` elements with the exception that they do not contain the list of descendant nodes. Each of these elements contains an additional attribute indicating the logical constant of the terminal node.

## 5.2. XSLT Template Style Sheets

XSLT (XSL Transformations) is an XML based language designed for specific purpose of conversion of documents in one XML based format to other formats. It has been introduced in 1999 in the form of W3 Consortium recommendation [33]. Although its primary aim is conversion between different forms of XML, XSLT is capable of converting XML documents even to non-XML file formats such as VHDL or, in this case, C++ syntax.

The XSLT is a declarative language. It does not state a program consisting of a deliberate sequence of instructions. Rather, XSLT defines a style sheet representing a collection of template rules which define the relationships between entities of the source and destination XML hierarchy. The XSLT processor analyzes the source XML document. It identifies the token elements and transforms

them according to the template rules in the corresponding XSLT document. New XML document is generated as an output. The order in which template rules are applied is not important and can be either sequential or recursive depending on the nature of the source XML document and the way it is parsed.

For details of development of XSLT style sheets we refer to [18] and [16].

The key stage in developing an XSLT style sheet for any particular transformation is identifying the correspondence between entities of the source and the target XML document formats.

Generation of branching programs in C/C++ syntax based on given decision diagrams is a multi stage process. Our system performs this operation by applying a series of XSLT templates to the elements of the original XML:DD document. The final output is a new document containing the code in C/C++. In order to achieve the optimal organization of the operation, these XSLT templates are grouped in three separate style sheets. The first two of them perform preprocessing, modifying the structure of original document. During this process a series of intermediary XML documents are produced. Only the last XSLT style sheet performs the actual mapping of XML elements to C/C++ syntax.

The recursive organization of XML:DD documents corresponds very well with the organization of code dictated by properties of common high-level programming languages. XML:DD documents are, as stated earlier, hierarchically structured. Each `<dd:node>` element contains in itself, nested other elements of the same type, nodes that belong are situated in its descending subdiagrams. Every such element corresponds with one basic If-Then-Else structure. In final output `<dd:node>` element is replaced with the following C/C++ code segment:

```
if(x1) then
{
  ...
}
else
{
  ...
}
```

This replacement process is repeated recursively for the descendants of the given node. Resulting C/C++ code will be nested in appropriate branches of If-Then-Else structure.

## 5.3. Examples

In Fig 4, we present the example of full branching code corresponding to the boolean function 2. We can note that the structure of the code follows closely the structure of the binary decision diagram generated for the same function.

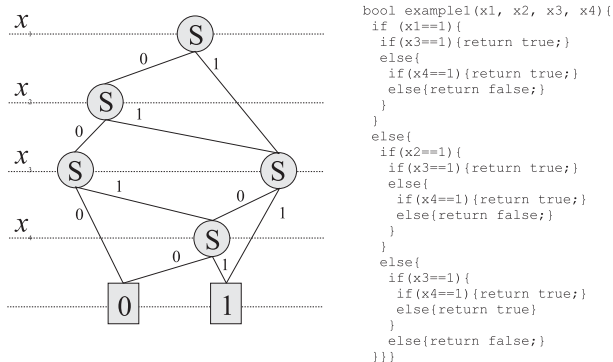


Figure 4. BDD and a branching program corresponding to the function  $f(x_1, x_2, x_3) = x_1x_2 \vee x_2x_3$ .

## 5.4. Generalization

The generalization that can be made to the method we presented in this paper is threefold.

So far we have limited ourselves to examining just the binary decision diagrams. This was done primarily for two reasons. Binary case is the most simple example of larger theory and thus easiest to follow. Second important factor is that, in this paper, we wanted to draw a comparison with the work done on automatic generation of hardware models in VHDL. This work was, due to complexities of hardware design focused on binary decision diagrams. However the proposed system is capable of generating C/C++ code even for non-binary, e.g. ternary or quaternary decision diagrams. There are no significant methodological differences in the way XSLT mechanism works in non-binary case. The main difference is that `<dd:node>` XML element corresponds is replaced with Switch-Case instead of If-Then-Else structure.

Second generalization direction is aimed at classes of decision diagrams derived using decision rules other than Shannon decomposition. In this case standard If-Then-Else branching structures need to be replaced with procedures performing operations equivalent to each particular decomposition rule.

Both of these modification can be easily combined, as in, for example, the case of Pseudo-Kronecker decision diagrams. Resulting program would be a semi-recursive hierarchy of boolean or arithmetic operations.

Finally since all of the needed programming structures, If-Then-Else, Switch-Case, recursiveness, etc. are present in a great majority of programming languages, and since the underlying algorithm remains unchanged, the proposed system can be adapted to produce its output in syntax of any formal language that satisfies these basic criteria.

## 6. CONCLUSION

Distinction between behavioral and structural models is one of the key points in high-level synthesis and modern hardware design. The XML base framework proposed in our previous papers has introduced a uniform form of rep-

resentation of different types of decision diagrams. As demonstrated in [30] this abstract form can be converted to VHDL based hardware models. These models are structural in nature. In this paper we have demonstrated how the same abstract form can be converted to branching programs, behavioral by definition. This ability to freely move from structural to behavioral type of models proves the versatility of proposed mechanism. Furthermore, this ability stems from one of the core properties of decision diagrams themselves. Decision diagrams are in essence a data structure, yet this data structure is used to represent a discrete function that implements some specific behavior. This inherent dualism is one of the factors that greatly increases the usability of decision diagrams as a concept. The need for a reliable standard format of their representation is thus even greater.

## 7. REFERENCES

- [1] Adelante Technologies, AIRT Builder, <http://www.adelantetechnologies.com>.
- [2] P. Ashar and S. Malik, "Fast functional simulation using branching programs.", ICCAD, 408-412, Oct. 1995.
- [3] P. Banerjee, "An overview of a compiler for mapping MATLAB programs onto FPGA", Proc. ASP-DAC 2003. Asia and South Pacific, Jan. 21–24 2003, Page(s):477 – 482
- [4] R. E. Bryant, "Graph-based algorithms for boolean functions manipulation", IEEE Trans. Comput., Vol. C-35, No. 8, 1986, 667 – 691.
- [5] K. Cagle, SVG programming: the graphical web, Apress, 8 July 2002.
- [6] Celoxica Corp, Handle C design language, <http://www.celoxica.com>.
- [7] CynApps Suite, Cynthesis application for higher level design, <http://www.cynapps.com>.
- [8] Esterel-C Language (ECL), Cadence website, <http://www.cadence.com>.
- [9] R. Drechsler, B. Becker, "Ordered Kronecker functional decision diagrams – a data structure for representation and manipulation of Boolean functions", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 10, 1998, 965 – 973.
- [10] G. DeMicheli, Synthesis and optimisation of digital circuits, McGraw Hill, 1994.
- [11] G. DeMicheli, D. Ku, F. Mailhot, T. Truong, The olympus synthesis system for digital design, IEEE Design & Test of Computers, 1990.

- [12] “Extensible Markup Language (XML) 1.0 (Third Edition)”, W3C Recommendation, <http://www.w3.org/TR/2004/REC-xml-20040204/>, 4 February 2004.
- [13] E. R. Harold, W. S. Means, XML in a nutshell, 2nd edition, O’Reilly, 2002.
- [14] H. M. Hasan Babu, T. Sasao, “Representations of multiple-output switching functions using multiple-valued pseudo-Kronecker decision diagrams”, Proc. 30th Int. Symp. on Multiple-Valued Logic, May 23 – 25, 2000, 147 – 152.
- [15] M. Haldar, A. Nayak, A. Choudhary, and P. Banerjee, “A system for synthesizing optimized FPGA hardware from Matlab”, Proc. International Conference on Computer Aided Design, San Jose, CA, November 2001.
- [16] S. Holzner, Inside XSLT, New Riders Publishing, 10 July 2001.
- [17] Y. Iguchi, T. Sasao, M. Matsuura, “Evaluation of multiple-output logic functions using decision diagrams”, Proc. ASP-DAC 2003., 312 – 315, 21–24 Jan., 2003.
- [18] S. Mangano, XSLT cookbook, O’Reilly Media Inc., December 2002.
- [19] D. M. Miller, R. Drechsler, “On the construction of multiple-valued decision diagrams”, Proc. 32nd Int. Symp on Multiple-Valued Logic, Boston, Massachusetts, USA, May 15 – 18 2002, 245 – 253.
- [20] S. Minato, Binary decision diagrams and applications for VLSI synthesis, Kluwer Academic Publishers, 1996.
- [21] S. Nagayama, T. Sasao, “Compact representations of logic functions using heterogeneous MDDs”, Proc. 33rd International Symposium on Multiple-Valued Logic, Tokyo, Japan, May 15 – 18, 2003, 247 – 252.
- [22] Overview of the Open systemC Initiative, SystemC website, <http://www.systemc.org>.
- [23] T. Sasao, J. T. Butler, “A design method for look-up table type FPGA by pseudo-Kronecker expansion”, Proc. 24th Int. Symp. on Multiple-Valued Logic, May 25 – 27, 1994, 97 – 106.
- [24] T. Sasao, M. Fujita, (ed.), Representations of discrete functions, Kluwer Academic Publishers, 1996.
- [25] F. Somenzi, “Efficient manipulation of decision diagrams”, Int. Journal on Software Tools for Technology Transfer, 3, 2001, 171 – 181.
- [26] F. Somenzi, “CUDD decision diagram package”, <http://bessie.colorado.edu/~fabio/CUDD>
- [27] R. S. Stanković, J. T. Astola, Spectral interpretation of decision diagrams, Springer, 2003.
- [28] R. S. Stanković, R. Drechsler, “Circuit design from Kronecker Galois field decision diagrams for multiple-valued functions”, Proc. 27th Int. Symp. on Multiple-Valued Logic, 275 – 280, May 28 – 30, 1997.
- [29] S. Stankovic, J. Astola, “XSLT based method for automatic generation of a graphical representation of a decision diagram represented using XML”, 7th International Workshop on Boolean Problems, Freiberg, Germany, 21 – 22 September 2006.
- [30] S. Stankovic, J. Takala, J. Astola, “Method for automatic generation of RTL in VHDL using decision diagrams”, Proc. The 2006 International TICSP Workshop on Spectral Methods and Multirate Signal Processing, SMMSP2006, 75–83, Florence, Italy, 2 – 3 September 2006.
- [31] S. Stojković, “UDDP universal decision diagram package”, Acta Electrotechnica et Informatica, No. 1, Vol 5, 2005, Košice, Slovakia.
- [32] System Compiler: Compiling ANSI C/C++ to synthesis-ready HDL. Whitepaper. C level Design Incorporated, <http://www.cleveldesign.com>
- [33] “XSL Transformations (XSLT) Version 1.0”, W3C Recommendation, <http://www.w3.org/TR/xslt>, 16 November 1999
- [34] S. Yang, “Logic synthesis and optimization benchmarks, version 3.0,” Tech. Report, Microelectronics Center of North Carolina, 1991.