

METHOD FOR AUTOMATIC GENERATION OF RTL IN VHDL USING DECISION DIAGRAMS

Stanislav Stanković¹, Jarmo Takala² and Jaakko Astola³

¹Tampere International Center for Signal Processing, Tampere University of Technology, P.O.Box 553, FIN-33101 Tampere, FINLAND, Stanislav.Stankovic@tut.fi

² Institute of Signal Processing, Tampere University of Technology, P.O.Box 553, FIN-33101 Tampere, FINLAND, Jarmo.Takala@tut.fi

³ Tampere International Center for Signal Processing, Tampere University of Technology, P.O.Box 553, FIN-33101 Tampere, FINLAND, Jaakko.Astola@tut.fi

ABSTRACT

In recent years, decision diagrams have earned a prominent place in the field of logic design as means of efficient representation of switching function, in terms of needed storage space and processing complexity, especially for the tasks of design, testing and verification of logic circuits. In this paper, we focus on a XML based system for description of the structure of decision diagrams with special attention on its application in high-level synthesis, automatization of design of logic networks.

The system we propose is capable of automatically producing hardware description, RTL models in VHDL. It uses XML for description of decision diagrams and a set of XSLT based scripts for conversion of decision diagrams to VHDL hardware model. VHDL is a well established industrial standard in this area and the RTL models produced by in this way can be synthesized using commercial logic synthesis tools.

This feature permits fast prototyping and much quicker experimentation regarding decision diagrams in the fields of logic synthesis and circuit testing. It also enables our framework to take the central role in much wider scope connecting various other software packages that deal with decision diagrams on some level with industrial standard simulation and synthesis software.

1. INTRODUCTION

Decision diagram is a data structure that permits efficient representation of discrete functions [3], [19], [23]. In our previous work, we have proposed the XML:DD, an XML based framework for description of the structure of decision diagrams. In this paper, we further extend this work to the problem of high-level synthesis i.e. automatic generation of hardware models in VHDL. We propose a system capable of producing automatically a VHDL description of a hardware implementation of a discrete function represented by a decision diagram.

The proposed solution analyzes the XML document that contains the description of the structure of a decision diagram. It applies a set of XSLT template style sheets to

identify structural elements of the diagram and generates the appropriate RTL model in VHDL.

This paper is organized in the following way. In Section 2, we discuss some examples of related work by other authors. We discuss the general issues regarding hardware implementation of discrete functions using decision diagrams in Section 3. In Section 4 we give a brief overview of important aspects of the XML:DD framework. In this section, we also establish the correspondence of the equivalent XML:DD and VHDL elements. In Section 5, we present hardware implementations of several standard benchmark functions produced using the proposed system. Finally, in Section 6, we give conclusions and closing remarks.

2. RELATED WORK

The problem of translating a high-level language behavioral or structural description into a register transfer level (RTL) representation is called high-level synthesis [9]. 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++ [7], [15]. There have been several commercial efforts to develop compilers taking C/C++ into VHDL or Verilog including CoWare, Adelante [1], Celoxica [5], C Level Design [30], and Cynapps [6]. SystemC developed by SystemC Consortium [21] 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 [2], [15].

In our work, we take a somewhat different approach focusing on structural descriptions of decision diagrams in the XML data description language. By taking this approach, we are able to avoid most of the problems regarding conversion of a behavioral model to structural model of an RTL since there is a well defined mapping of deci-

sion diagram structural entities to hardware elements. We are able to take this approach because most of the software tools dealing with decision diagrams, which are to serve as a front end to the proposed system, focus on the problems of structural optimization of decision diagrams. In the following section, we discuss the issues of hardware implementation of discrete functions using decision diagrams.

3. HARDWARE IMPLANTATION OF DISCRETE FUNCTIONS USING DECISION DIAGRAMS

Decision diagrams are obtained by reduction of decision trees [28]. Decision trees are structures obtained by a recursive application of a set of expansion rules to a discrete function. Different types of decision diagrams are defined according to the choice of used expansion rules [28]. In this paper we focus exclusively on binary decision diagrams, a class of decision diagrams based on Boolean logic. Possible expansion rules in binary decision diagrams include the following:

1. Shannon, $f = \bar{x}_1 f_0 \oplus x_1 f_1$,
2. Positive Davio, $f = f_0 \oplus x_1 f_2$,
3. Negative Davio, $f = f_1 \oplus \bar{x}_1 f_2$.

Fig. 1 represents logic circuits realizing each of these three expansions. Each node in the decision diagram thus corresponds to a hardware unit realizing the given expansion. Hardware realizing a switching function using a decision diagram is a network of interconnected units performing the given expansion.

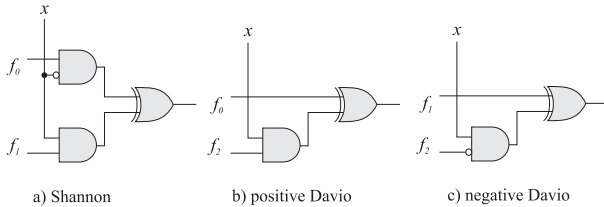


Figure 1. Logic circuits corresponding to three types of expansions, $f_2 = f_0 \oplus f_1$.

The choice of expansion rules significantly influences the structure of a decision diagram. In order to minimize the size of the decision diagram, different decision rules can be used within the same diagram. In the case of Kronecker decision diagrams, a single expansion rule is assigned to all the nodes on the same level of the diagram. If the decomposition rule is selected freely at each node, the decision diagram a Pseudo Kronecker decision diagram. For further information about Kronecker decision diagrams we refer to [8], [11] and the reference there in. Detailed description about the Pseudo Kronecker decision diagrams and their application for representation of switching functions can be found for instance in [14] and other related works by the same authors.

The system we propose, is capable of generating hardware architectures for all of these three classes of 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 consists of a library of basic components realizing three different expansions and 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, $f(x_1, x_2, x_3) = x_1 \bar{x}_2 \vee x_2 x_3$, using the corresponding BDD can be seen in Fig 2.

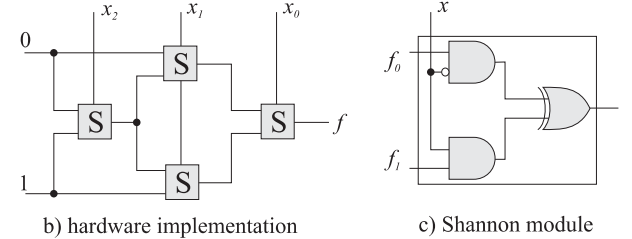
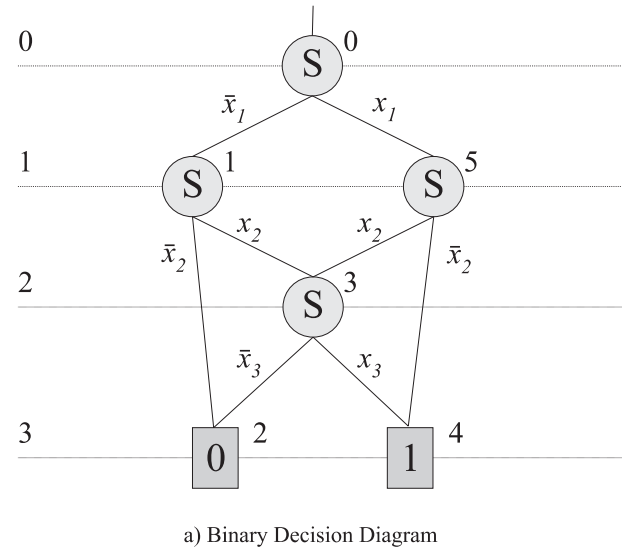


Figure 2. BDD and a logic network corresponding to the function $f(x_1, x_2, x_3) = x_1 \bar{x}_2 \vee x_2 x_3$.

3.1. Decision Diagrams using VHDL

We assume that the reader is already familiar with the basic concepts of high level hardware languages in general and VHDL, in particular. There is a wide variety of books on this subject such as [22].

Although VHDL is a very complex language that permits several different approaches to hardware design, all VHDL hardware designs produced by the proposed system share a common form, due to the fact that all binary decision diagrams have the same rigid basic structure.

The design consists of one top-level entity and its associated structural architecture. Since we deal with bi-

nary decision diagrams, the number of incoming ports is determined by the number of logic variables of the implemented switching function. Two additional input ports are provided for the logical constants. Since we have limited ourselves to dealing only with single output decision diagrams the top-level entity has only one output port.

Separate entities representing hardware implementations of each of three standard decomposition rules are provided. References to these entities will be created appropriately for each node of a decision diagram in the architecture section of the design. The first part of the architecture contains declarations of all the external entities used in the architecture description. In our case, the internal signals of the hardware architecture correspond to the edges of the decision diagram. Since we are dealing with binary decision diagrams, the `std_logic` type is used for ports of entities and internal signals of the design.

Therefore, the basic outline of a VHDL design looks as follows:

```

ENTITY entity_name IS
  PORT (list_of_input_ports : IN std_logic;
output_port : OUT std_logic);
END ENTITY entity_name;
ARCHITECTURE arch_name OF entity_name IS

...declarations of components as needed...

COMPONENT my_shannon
  PORT (f0, f1, x : IN std_logic; z : OUT std_logic);
END COMPONENT;
COMPONENT my_pdavio
  PORT (f0, f1, x : IN std_logic; z : OUT std_logic);
END COMPONENT;
COMPONENT my_mdavio
  PORT (f0, f1, x : IN std_logic; z : OUT std_logic);
END COMPONENT;

SIGNAL internal_signals : std_logic;
BEGIN

...actual structure of the decision diagram...

END arch_name;

```

4. XSLT BASED SOLUTION

4.1. Basic Notions on XML

XML [12], [13] 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, the XML 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.

4.2. XML based framework for Decision Diagrams

The structure of XML:DD documents containing decision diagrams is complex. The focus in the 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.

An XML document store data in the form of the structured text. Elements of the text can contain other nested elements forming a complex hierarchy. In the case of the XML:DD, the root element of the hierarchy is the `<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.

Then nodes of a decision diagram are represented in the form of `<dd:node>` elements. Each node contains another `<dd:node>` sub-element corresponding to the next node in an 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 nodes contain attributes declaring the unique identifier of each node, the level to which the node belongs, and an expansion (decomposition rule) that was applied at every particular node. An additional attribute indicating the logical constant is associated with each terminal node. Edges of the decision diagrams are stored in the XML:DD framework in form of lists of descendants and parents associated with each particular node using the `<dd:children>` and the `<dd:parents>` nested elements respectively. The XML code describing one particular node of a decision diagram has the following general form:

```

<dd:node terminal='0' id='5' level='1' rule='Shannon'>
  <dd:parents point='0' />
  <dd:children point='3'>
    <dd:next_child point='4' />
  </dd:children>
</dd:node>

```

Fig. 3 shows an example of binary decision diagram and data structures which XML:DD uses to store it.

As we can see, the XML:DD framework makes a heavy use of recursive properties of XML. Although this matches well the nature of decision diagrams, this organization of data structures is not well suited for conversion to VHDL. Furthermore part of the information stored in this way is not needed for this particular application.

4.3. XSLT Template Style Sheets

The 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 [32]. Although its primary aim is conversion between different forms of XML, the XSLT is capable of converting XML documents even to non-XML

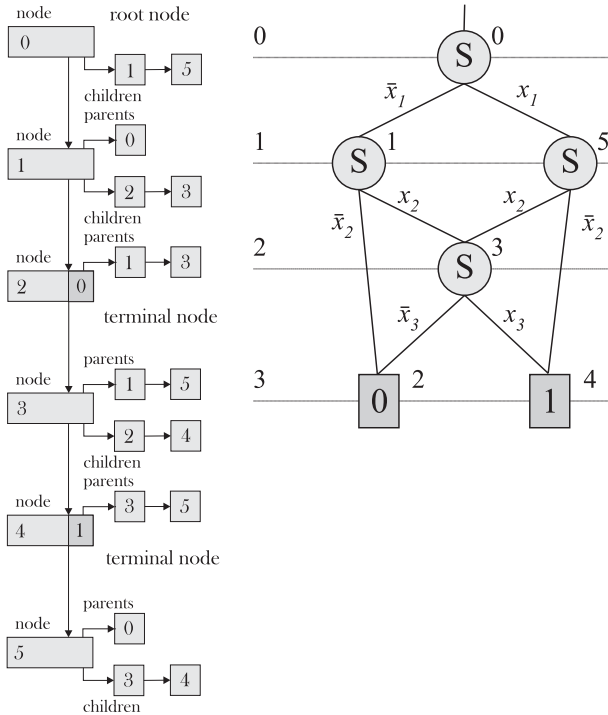


Figure 3. XML:DD data structures

file formats such as VHDL.

The XSLT is a declarative language. It does not state a program consisting of a deliberate sequence of instructions. Rather, the XSLT defines a style sheet representing a collection of template rules which define the relationships between entities of the source and the 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 [16] and [17].

The key stage in developing an XSLT style sheet for any particular transformation is the identification of correspondence between entities of the source and the target XML document formats. The process of producing a VHDL hardware description from XML:DD will, consist of two distinct steps:

1. Preprocessing
2. High-level synthesis

In what follows we give a detailed description of both of these steps.

4.4. Preprocessing

In the first phase of the process, we apply one preprocessing XSLT style sheet to produce a more suitable version of

the XML representation of a decision diagram. This style sheet produces a simple XML document containing only a list of decision diagram nodes without any nested elements. Each node contains a simple list of its descendants. The attributes indicating the ID, level and expansion rule are preserved.

There is no logical operation performed in terminal nodes of decision diagrams. Terminal nodes simply indicate values of the logical constants. Therefore, terminal and non-terminal nodes will assume different roles in the RTL model. This difference is clearly indicated in the preprocessed XML document by storing of the terminal nodes in a separate list. All the recursive aspects of the original XML:DD document are removed by the preprocessing style sheet.

4.5. High-Level Synthesis

The second step of the process represents the final conversion of an intermediate XML form to VHDL by using a separate XML style sheet.

In our case, the XSLT style sheet first identifies `<dd:tree>`, the root element of our XML hierarchy. This is a wrapper element and it corresponds to the general template of a VHDL document. A skeleton of a VHDL document is created at this stage including the header with the declaration of a new entity. The name `my_DD` is given to this entity by default.

The XSLT style sheet generates a port list of `my_BDD` entity automatically. Ports with names `con*` (where `*` denotes a numeral) represent inputs for logical constants. Since we are dealing exclusively with binary logic, there is only two of them `con1` and `con0`. Logical variables of the discrete functions are mapped to ports with names of form `x*`. Their number is determined by the number of levels in the decision diagram. One output port is assigned, with default name `'o'`. All declared ports are of type `'std_logic'`.

Next, the XSTL style sheet generates a declaration of architecture and associates it with the declared entity. Name `my_dd_arch` is assigned by default to this architecture. Depending on the value of the `'type'` attribute in tree element declarations for the appropriate components will be included in the VHDL document. Either a single `my_shannon`, `my_pdavio` or `my_ndavio` component or the combination of components will be included in the declaration.

Edges of the decision diagram correspond to the internal signals of the VHDL architecture. These signals are declared in the header of the architecture declaration. The number of the internal signals is equivalent to the number of non-terminal nodes excluding the root node. The names assigned to these signals are of the form `o*`, where `*` denotes a numeral.

What follows is the body of the hardware architecture with the list of component instances representing the nodes. The component type is selected according to the value of the `'rule'` attribute of the node element. Port map is generated for each component instance by analyz-

ing the list of its children. Each component has two inputs and one output of type 'std_logic'.

Appropriate register entities are added to the design to facilitate the connections of the described unit with other modules. The register entities can be a part of a pre-generated module library or generated on the fly by using XSLT. The XSLT can be used to automatically produce a test bench for the RTL model generated in this way.

5. EXAMPLES

To test the validity of the proposed concept, we have used the system to automatically produce RTL descriptions in VHDL of hardware implementations of a number of binary functions. We have focused on functions from MCNC (Microelectronics Center of North Carolina) set of logic design benchmark functions. We produced a number of different binary decision diagrams for each function. The binary decision diagrams were converted to valid XML:DD documents. For this task, we have used software created by Suzana Stojiljković at Faculty of Electronics at University of Niš, Niš, Serbia, in conjunction with an export module capable of producing the XML:DD documents. We have used these XML:DD documents as an input to the proposed system, to produce hardware descriptions in VHDL. We have made the final RTL synthesis of the produced VHDL code for various FPGA technologies, using the Mentor Graphic LeonardoSpectrum synthesis tool.

In this section, we present the results obtained for five representative functions selected from the benchmark library. For each function, we have generated four different binary decision diagrams:

1. Binary decision diagram with the Shannon derivative at each level,
2. Binary decision diagram with the positive Davio derivative at each level,
3. Kronecker binary decision diagram with the positive Davio applied at the topmost level and the Shannon derivative in all other levels,
4. Kronecker binary decision diagram with the positive Davio applied at the topmost and lowest level and the Shannon derivative in all other levels.

We have tested the examples for two FPGA families from two different companies, Stratix III family from Altera, and Spartan 3 from Xilinx. The first set of tables represents the results of implementation of selected functions by using Spartan 3 product family.

The second set of tables corresponds to the results obtained for Stratix III FPGA technology from Altera.

To better illustrate these results on Fig. 4, we present a binary decision diagram for the function `misex1_5` from the benchmark set generated by recursive application of the Shannon derivative and a RTL schematic of its hardware implementation in VHDL. Please notice that logic variables `x 4` and `x 7` do not have any influence on the function of the circuit. This is due to the fact that `misex1_5`

Table 1. Selected benchmark functions implemented using a Shannon binary decision diagram and Xilinx Spartan 3 FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
<code>misex1_5</code>	8	163.6	0.1	24
<code>alu4_0</code>	14	163.6	0.1	38
<code>cordic_0</code>	23	83.1	0.5	93
<code>apex2_0</code>	39	40.2	0.2	3006
<code>apex1_0</code>	45	122.9	0.2	56

Table 2. Selected benchmark functions implemented using a binary decision diagram with positive Davio derivative and Xilinx Spartan 3 FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
<code>misex1_5</code>	8	163.6	0.1	27
<code>alu4_0</code>	14	143.0	0.2	66
<code>cordic_0</code>	23	163.6	0.1	71
<code>apex2_0</code>	39	130.4	0.14	54
<code>apex1_0</code>	45	146.6	0.3	39

Table 3. Selected benchmark functions implemented using a Kronecker binary decision diagram and Xilinx Spartan 3 FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
<code>misex1_5</code>	8	163.6	0.1	31
<code>alu4_0</code>	14	129.5	0.3	77
<code>cordic_0</code>	23	83.1	0.5	93
<code>apex2_0</code>	39	40.2	0.2	3006
<code>apex1_0</code>	45	122.9	0.2	56

Table 4. Selected benchmark functions implemented using a Kronecker binary decision diagram with different choice of decomposition rules and Xilinx Spartan 3 FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
<code>misex1_5</code>	8	163.6	0.1	26
<code>alu4_0</code>	14	129.5	0.3	77
<code>cordic_0</code>	23	83.0	0.4	93
<code>apex2_0</code>	39	40.2	0.2	3006
<code>apex1_0</code>	45	120.7	0.1	53

Table 5. Selected benchmark functions implemented using a Shannon binary decision diagram and Altera Stratix III FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
misex1_0	8	392.2	0.0	17
alu4_0	14	175.0	0.2	50
cordic_0	23	116.3	0.1	65
apex2_0	39	46.2	0.6	1625
apex1_0	45	43.8	0.4	39

Table 6. Selected benchmark functions implemented using a positive Davio binary decision diagram and Altera Stratix III FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
misex1_0	8	392.2	0.0	17
alu4_0	14	151.4	0.1	48
cordic_0	23	149.7	0.2	60
apex2_0	39	166.6	0.1	43
apex1_0	45	210.8	0.0	33

Table 7. Selected benchmark functions implemented using a Kronecker binary decision diagram and Altera Stratix III FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
misex1_0	8	392.2	0.0	17
alu4_0	14	167.0	0.1	53
cordic_0	23	114.6	0.0	65
apex2_0	39	43.5	0.3	39
apex1_0	45	173.5	0.0	44

Table 8. Selected benchmark functions implemented using a Kronecker binary decision diagram with different choice of decomposition rules and Altera Stratix III FPGA technology.

function	num. of inputs	max. freq.	slack	acc. inst.
misex1_0	8	392.2	0.0	17
alu4_0	14	167.0	0.1	53
cordic_0	23	106.0	0.1	65
apex2_0	39	43.5	0.3	39
apex1_0	45	215.1	0.0	38

is the fifth output of the multi output `misex1` function. In Fig. 5 we show a binary decision diagram created using the positive Davio decomposition for the same switching function and the corresponding RTL schematic. The evident simpler structure of this circuit clearly indicates the advantage of the choice of a different decomposition rule.

6. CONCLUSION

In this paper, we have introduced an XML based framework for representation of decision diagrams for switching functions. We have demonstrated the capability of the proposed solution to represent various forms of decision diagrams in way that is convenient for further applications. Further more we have demonstrated this property on the problem of automatic generation of RTL models using VHDL syntax. To illustrate this, we have presented a set of examples, binary decision diagrams produced using different decomposition rules and their hardware implementations using two different FPGA families. For these examples we have used a set of standard benchmark functions for logic design provided by NCSC.

The demonstrated properties of the proposed system, its flexibility and robustness, justifies the belief that it can be a valuable addition to the existing set of tools for the logic design.

7. REFERENCES

- [1] Adelante Technologies, AIRT Builder, <http://www.adelantetechnologies.com>.
- [2] 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
- [3] R. E. Bryant, "Graph-based algorithms for Boolean functions manipulation", IEEE Trans. Comput., Vol. C-35, No. 8, 1986, 667 – 691.
- [4] K. Cagle, SVG Programming: The Graphical Web, Apress, 8 July 2002.
- [5] Celoxica Corp, Handle C Design Language, <http://www.celoxica.com>.
- [6] CynApps Suite, Cynthesis Application for Higher Level Design, <http://www.cynapps.com>.
- [7] Esterel-C Language (ECL), Cadence website, <http://www.cadence.com>.
- [8] R. Drechsler, B. Becker, "OKFDDs-algorithms, applications and extensions", in T. Sasao, M. Fujita, (ed.), Representations of Discrete Functions, Kluwer Academic Publisher, Boston, 1996, 163 – 190.
- [9] G. DeMicheli, Synthesis and Optimisation of Digital Circuits, McGraw Hill, 1994.

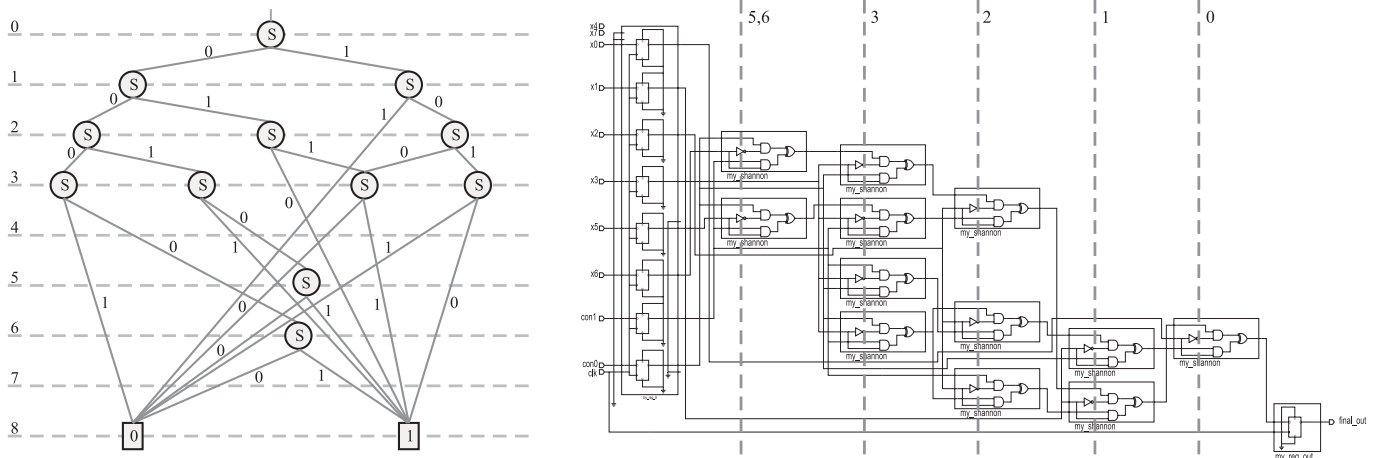


Figure 4. Decision diagram created using a Shannon decomposition rule and RTL schematic for the fifth output of misex1 benchmark function

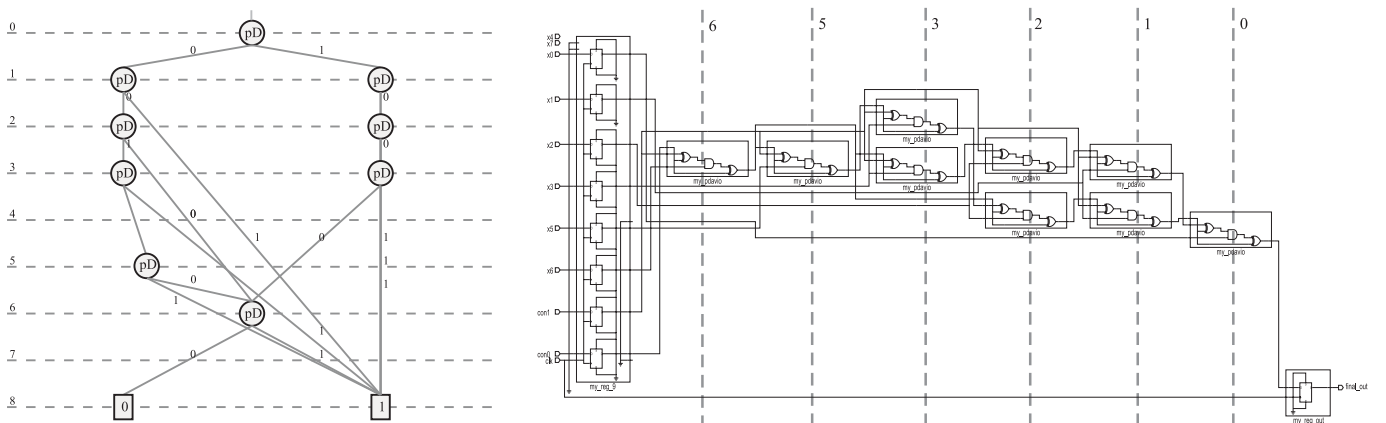


Figure 5. Decision diagram created using a positive Davio decomposition rule and RTL schematic for the fifth output of misex1 benchmark function

- [10] G. DeMicheli, D. Ku, F. Mailhot, T. Truong, *The Olympus Synthesis System for Digital Design*, IEEE Design & Test of Computers, 1990.
- [11] 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.
- [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] S. Mangano, *XSLT Cookbook*, O'Reilly Media Inc., December 2002.
- [18] 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.
- [19] S. Minato, *Binary Decision Diagrams and Applications for VLSI Synthesis*, Kluwer Academic Publishers, 1996.
- [20] 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.
- [21] Overview of the Open SystemC Initiative, SystemC website, <http://www.systemc.org>.
- [22] D. L. Perry, *VHDL: Programming by Examples*, McGraw-Hill Professional, 2002.
- [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] "Scalable Vector Graphics (SVG) 1.1 Specification", W3C Recommendation, <http://www.w3.org/TR/SVG11>, 14 January 2003.
- [26] F. Somenzi, "Efficient manipulation of decision diagrams", *Int. Journal on Software Tools for Technology Transfer*, 3, 2001, 171 – 181.
- [27] F. Somenzi, "CUDD Decision Diagram Package", <http://bessie.colorado.edu/~fabio/CUDD>
- [28] R. S. Stanković, J. T. Astola, *Spectral Interpretation of Decision Diagrams*, Springer, 2003.
- [29] 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*, May 28 – 30, 275 – 280, 1997.
- [30] *System Compiler: Compiling ANSI C/C++ to Synthesis-ready HDL*. Whitepaper. C level Design Incorporated, <http://www.cleveldesign.com>
- [31] A. H. Watt, C Lilley, *SVG Unleashed*, Sams, 20 September 2002.
- [32] "XSL Transformations (XSLT) Version 1.0", W3C Recommendation, <http://www.w3.org/TR/xslt>, 16 November 1999
- [33] S. Stojković, "UDDP Universal Decision Diagram Package", *Acta Electrotechnica et Informatica*, No. 1, Vol 5, 2005, Košice, Slovakia.
- [34] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," Tech. Report, Microelectronics Center of North Carolina, 1991.