DECISION DIAGRAMS AND PASS TRANSISTOR LOGIC SYNTHESIS

V. Bertacco
S. Minato
P. Verplaetse
L. Benini
G. De Micheli

Technical Report No.: CSL-TR-97-748

December 1997

This research was sponsored in part by ARPA, under grant No. DABT63-95-C-0049.

Decision Diagrams and Pass Transistor Logic Synthesis

V. Bertacco S. Minato P. Verplaetse L. Benini G. De Micheli

CSL-TR-97-748

December 1997

COMPUTER SYSTEMS LABORATORY
Departments of Electrical Engineering and Computer Science
Gates Computer Science Building, #408
Stanford University
Stanford California 94305-9040

Abstract

Since the relative importance of interconnections increases as feature size decreases, standard-cell based synthesis becomes less effective when deep-submicron technologies become available. Intra-cell connectivity can be decreased by the use of macro-cells. In this work we present methods for the automatic generation of macro-cells using pass transistors and domino logic. The synthesis of these cells is based on BDD and ZBDD representations of the logic functions. We address specific problems associated with the BDD approach (level degradation, long paths) and the ZBDD approach (sneak paths, charge sharing, long paths). We compare performance of the macro-cells approach versus the conventional standard-cell approach based on accurate electrical simulation. This shows that the macro-cells perform well up to a certain complexity of the logic function. Functions of high complexity must be decomposed into smaller logic blocks that can directly be mapped to macro-cells.

Keywords & Phrases: logic synthesis, technology mapping, pass transistor logic, domino logic, BDD, ZBDD, low power design.

1 Introduction and motivation

Logic synthesis has been an enabling technology for the design of Very Large Scale Integration (VLSI) circuits and systems. The logic abstraction of a circuit is usually in terms of logic gates (later mapped into cells). Most current synthesis tools assume that the key cost metrics (i.e., delay, area and power) depend mainly on the cells, and model interconnect as a parasitic effect affecting the cell performance (i.e., delay and power).

As deep-submicron technologies become available, the cell-based design style loses modeling precision: active cell area shrinks and the relative importance of interconnections increases. The cost metrics are dominated by parasitic effects. Therefore it is increasingly difficult to optimize interconnect-dominated circuits at the logic level of abstraction.

In this work we investigate a design style that exploit automatically generated macro-cells based on pass transistors. Such macro-cells may have a larger size than usual semi-custom cells, and their cost parameters can be inferred from the transistor topology. The use of macro-cells gives us two advantages: first, the number of primitive elements to be instantiated and connected decreases. Second, the relative importance of interconnection between such blocks decreases as well. Consequently logic synthesis is at least partially relieved from the burden of managing a huge number of instances of small atomic primitives, and from some of the uncertainty caused by the high relative impact of wiring-related costs.

We want also to have some insight in the internal connecting structure of the macro-cells. To this end we leverage the close affinity between abstract structures known as Binary Decision Diagrams (BDDs) and Zero-suppressed Binary Decision Diagrams (ZBDDs) and transistor-level implementations based on pass transistor logic.

Several approaches to direct synthesis of pass transistor networks have been proposed. In their pioneering work [1], Yano et al. proposed a BDD-based pass transistor synthesis tool. More recent work [2] proposed FBDD-based pass transistor synthesis, but no experimental results on transistor-level implementations are provided. Konishi et al. in [3] proposed a BDD-based synthesis tool for low-power pass transistor logic.

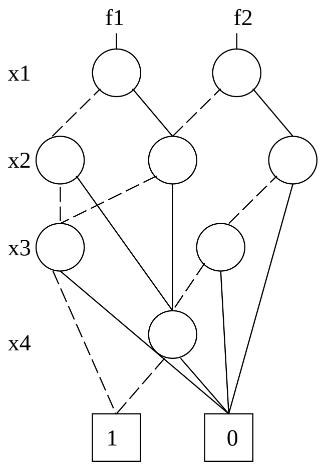
Differently from the work in [1][2][3] we focus on synthesizing one logic block at a time, and we do not assume that BDDs should be used to represent the complete specification. No weak pull-up devices are used to restore the level of degraded signals, instead the use of an asymmetric threshold technology to mitigate level degradation is being explored. Based on accurate electrical simulation the relative performance of several alternative implementation styles are compared.

2 BDD-based synthesis

Given a multi-output Boolean function of n inputs, $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \cdots, f_{n_o}(\mathbf{x})]$, we can represent $\mathbf{f}$ with a multi-rooted BDD, as shown in Figure 1 (a). The BDD is levelized and there is a one-to-one correspondence between levels and input variables.

We can map the BDD of $\mathbf{f}$ to a pass transistor circuit with a straightforward transformation. BDD nodes are mapped to pass transistor multiplexers, as shown in Figure 1 (b).

We use only NMOS transistors in the pass transistor network. Advantages of this choice is that the input load capacitance is minimized (faster circuits, and lower power dissipation), and that



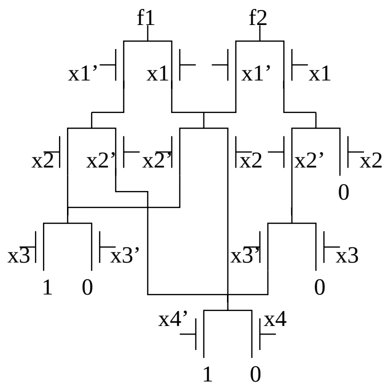


Figure 1: Multi-rooted BDD and its pass transistor mapping

the area is reduced as well. However, we need both input polarities to drive the pass transistor multiplexer.

2.1 Asymmetric thresholds

Unfortunately, NMOS devices have poor driving characteristics for the "high" logic value: they turn off as V_{ds} (drain-source voltage) gets close to V_t. As a result raising transition are remarkably slower than falling transitions on the pass gate nodes. The full output swing can be restored by inserting a CMOS buffer on each output of the pass transistor network. Even though V_{t0} (the zero bias threshold voltage) is usually bigger (in absolute value) for PMOS devices than for NMOS devices, the actual voltage degradation can be close to or even surpass V_{t0_P} due to the body effect. When the buffer is driven by a degraded high voltage the PMOS transistor will not be fully off, and there will be some leakage current through the buffer.

A first solution to this problem is to use a weak pull-up devices to restore the degraded value [1][3], as is shown in Figure 2 (a). Note that this only decreases the power dissipation, but does not improve the noise margin. The resulting logic is ratioed, which makes it hard to generate longer chains of pass gates with optimal sizing (often minimal). A more advanced solution involves adopting *asymmetric thresholds* (Figure 2 (b)). There are two types of NMOS transistors: for the static CMOS gates we use transistors with the usual thresholds, but for the pass gates NMOS transistors with reduced threshold (usually close to 0) are applied. This technology adjustment reduces the leakage problem on the buffers at no area cost and increases the speed of the rising transitions. An extra mask and extra ion-implantation step is required during fabrication to adjust the threshold for the NMOS devices.

2.2 Buffer insertion

Another problem with the pass transistor network is the presence of long paths: the delay of a chain of n pass transistors is proportional to n^2. The path length can be reduced by inserting buffers,

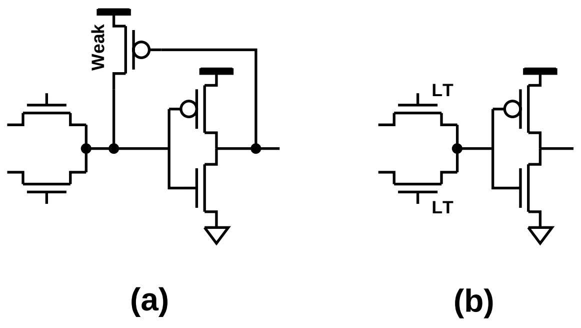


Figure 2: (a) Weak pull-up (b) Asymmetric thresholds

but this increases area. The optimum path length depends on the delay of the pass transistors and the buffer delay, as well as the speed-area trade-off point that one wants to obtain.

When a BDD with complemented edges [4] is mapped to a pass transistor circuit, each complemented edge corresponds to an inverter, which acts as an *implicit* buffer. Unfortunately, we cannot guarantee that implicit buffers solve all long-path problems, and *explicit* buffers must be inserted as well. Algorithms similar to those in [3] can be applied. One more type of buffering is required. When mapping large BDDs, some nodes may have a large number of ancestors. In the pass transistor network, this translates into a large node capacitance on the output of the corresponding multiplexer. This effect is mitigated by buffering high fanout nodes.

In the current implementation the buffer insertion procedure first traverses the BDD once and inserts buffers on the output of nodes with fanout larger than FO_{max}. Then, the node traversal is repeated and additional buffers are inserted such that the maximal unbuffered path length is L_{max}. The complexity of the buffering algorithm is $O(N_{nodes})$.

Buffering can improve the performance of the circuit, but other transformations have a strong influence on the quality of the final pass transistor implementation. It is known that the complexity of BDDs is very sensitive to the ordering of the input variables. In the current implementation, we perform variable reordering based on sifting [5] and we select for mapping the order that produces the BDD with the smallest number of nodes. This choice is based on the observation that node count has a high correlation with number of transistors (although there is some uncertainty due to buffering) and some correlation with the speed of the circuit.

3 ZBDD-Based synthesis

Zero-suppressed BDDs [6] are well-suited for representing Boolean functions in sum-of-products (SOP) form. Figure 3 (a) shows a SOP representation of a function and its ZBDD. Notice that this representation labels nodes with literals instead of variables. Thus x and x' nodes can both be present in the same ZBDD.

An important property of ZBDDs is that they allow sharing of subgraphs. Subgraph sharing has a precise meaning when ZBDDs are used to represent SOPs: it is equivalent to *factoring* [7]. Intuitively, ZBDDs do not only represent Boolean functions, but also their factored forms. This is a paramount property in the application of ZBDDs to direct mapping of Boolean functions to transistor-level net lists.

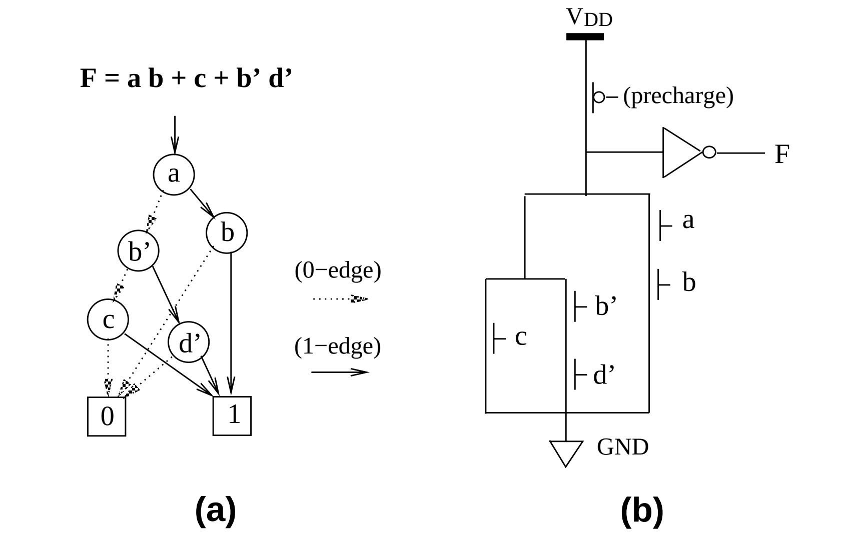


Figure 3: (a) A ZBDD representing a cube set (b) Corresponding ZBDD-based mapping

3.1 Mapping ZBDDs into Domino Logic

The logic functionality of a ZBDD node is $f_d + xf_x$, where f_d (0-edge) represents the subset of f which does not depend on x, and where f_x (1-edge) is the cofactor of f w.r.t. to x. Implementation of such a function in pass transistor logic requires a single transistor, as shown in Figure 4. The 1-edge connection is realized with a transistor, the 0-edge is just a wire.

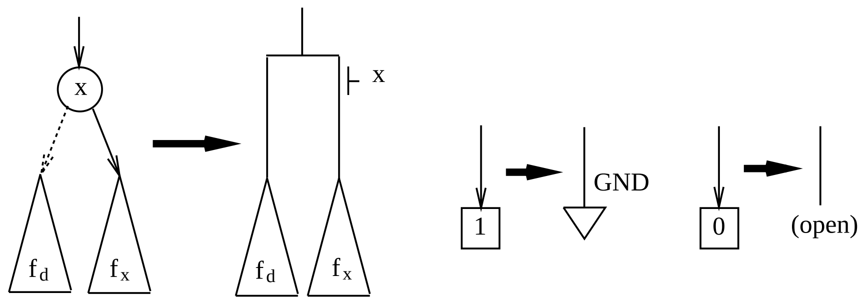


Figure 4: Basic rules of ZBDD-based mapping

As a consequence, mapping a ZBDD to pass transistors produces a switch network. We cannot use the network to drive both "high" and "low" values to the output, as in the pass transistor approach of Section 2. Instead, only the "low" values are driven, which makes the network ideal for dynamic logic families such as the well-known *domino* logic [8]. All output nodes are precharged during one phase of the clock and conditionally discharged by a switch network during the opposite phase (the two phases are known as *precharge* and *evaluation*). Based on these observations it is easy to conclude that ZBDDs are well suited to be directly mapped into domino logic (Figure 3 (b)).

NMOS transistors implement the switch networks in domino logic because they have smaller effective resistance than PMOS transistors. Since NMOS transistors drive strongly and without voltage degradation the "low" logic value, the natural choice is to assume that the sink of the switch network is always connected to GND.

The advantages of the ZBDD mapping to domino logic are: (i) the exploitation of the high speed and low input load of domino logic, (ii) the possibility of leveraging traditional logic minimization tools to reduce the number of literals in a SOP representation [9] that directly translates to reducing the number of transistors, (iii) the sharing of common sub-factors that further reduces the transistor count.

3.2 Sneak paths

The mapping of ZBDDs to transistors is not completely straightforward, mainly because of the possible unwanted conductive paths (*sneak paths*) in the switch network due to the bidirectional behavior of transistors (Figure 5).

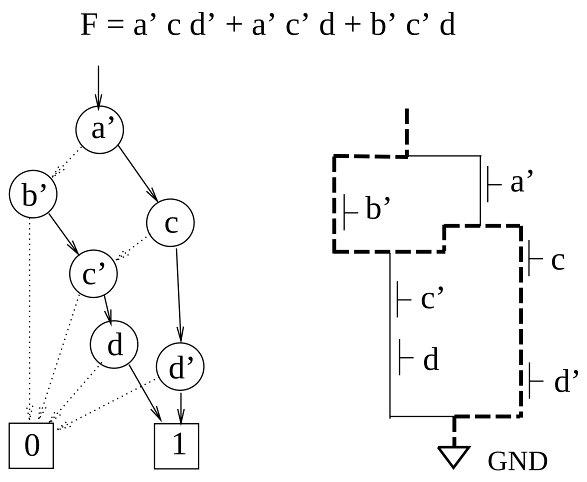


Figure 5: Sneak path problem

To avoid wrong direction flows, we insert buffers on each multiple fan-out node. Buffer insertion is shown in Figure 6. Buffer insertion increases the number of transistors and affect the delay. However, the delay can sometimes be improved because buffering reduces the length of long transistor chains. It is important to note that in this case buffering is required for functional correctness and not for performance enhancement like in the BDD case.

There are some cases where we can omit buffer insertion in a multiple-fanout node. For example, the circuit shown in Figure 7 (a) contains two fanout nodes, but there are no sneak paths. As shown in Figure 7 (b), when there is no bypath of the fanout node P, we do not have to insert a buffer at P because all the paths from the root node F to GND should pass through P in the same direction. This condition can be checked simply by the formula $F\%P = 0$, where % means the remainder of the algebraic division of the cube sets F and P. This formula indicates that the cube set P is exactly a factor of F, in other words, every cubes in F includes one of the cube in P. So, every path of F should pass through the node P. Notice that algebraic division can be executed quickly exploiting recursive ZBDD manipulation [7]. If the circuit has multiple output functions, we can omit buffer insertion if the formula is satisfied for all the output functions which are relevant to the node P.

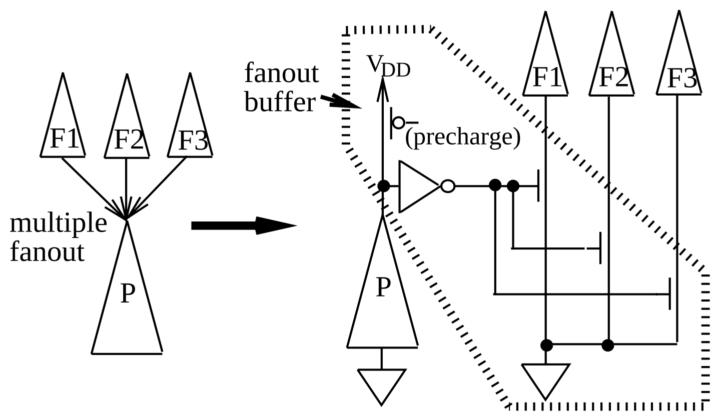


Figure 6: Buffer insertion

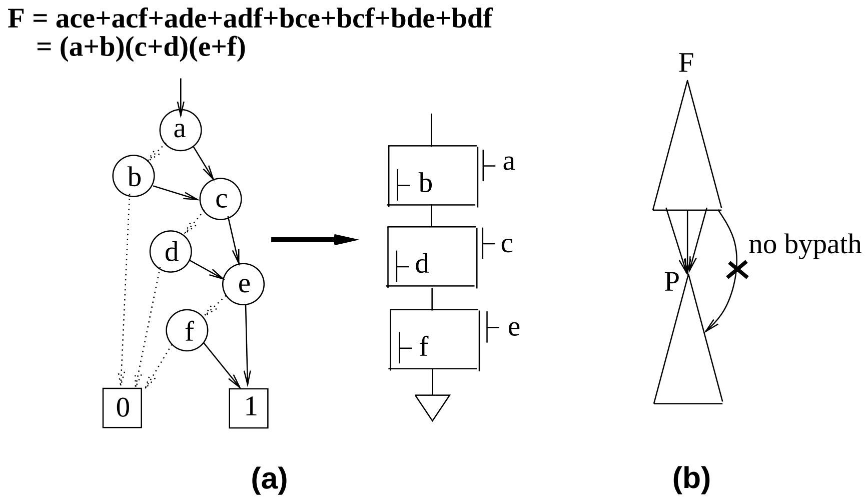


Figure 7: (a) An example when no buffer is needed (b) Condition for buffer elimination

The above condition does not cover all the cases where we can omit buffers. However, testing the additional cases is more involved and the current implementation of our mapper does not support advanced buffer elimination tests.

After checking the buffer insertion condition, even if there no multiple fanout nodes, we force buffer insertion to split excessively long chains of transistors. Finally, in order to minimize charge sharing problems, we insert internal precharge transistors every L_{pre} transistors in series.

As for BDDs, variable ordering has a strong impact on the size of ZBDDs. However, in the current implementation we use the same ordering for both decision diagrams.

4 Experimental results

We have implemented the BDD and ZBDD mapping procedure described in the previous sections and tested them on a small set of benchmarks. Macro-cells with 8–50 inputs and 1–40 outputs are used. Note that most current semi-custom library cells have 2–20 transistors and that 90%

of the cells have 1–8 inputs and a single output. For BDD mapping we used a BDD package developed at University of Padova, Italy, while ZBDD mapping was built on top of CUDD [11]. Both mapping tools read `slif` files and produce SPICE net lists. The ZBDD mapping procedure uses ESPRESSO [9] as a preliminary optimization step before mapping. Accurate circuit level simulation was performed using HSPICE. We used delay estimation to generate input patterns for the BDDs that would reveal the true maximum delay of the circuits. Delay estimation is also useful for critical path analysis, and can be used in buffer insertion algorithms.

In order to get an idea on the relative performance of BDD- and ZBDD-based circuits we compared the performance with a commercially available tool using a simple standard-cell containing 12 basic gated primitives and buffers. The net list generated was replaced by the transistor-level implementation (static CMOS) of each gate. We used the same CMOS technology for all the transistor-level implementations (a 0.5μ CMOS technology). We used asymmetric thresholds for the BDD-based implementation, once with minimum sized and once with double sized pass transistors. The values for L_{max} and FO_{max} are 6 and 4 respectively. Charge sharing is prevented by using a conservative internal precharge scheme: $L_{pre} = 2$.

Bench	Inputs	Outputs	Muxes	Delay (ns)		
				DT	DT2	Std
b1	3	4	10	0.667	0.628	0.487
daio	5	6	15	0.681	0.679	0.811
cm150a	21	1	32	0.835	0.903	1.049
s208	19	10	76	1.477	1.497	1.426
b9	41	21	157	2.004	1.789	1.187
s832	23	24	286	3.038	3.575	2.374
alu4	14	8	472	3.903	4.549	3.243
s1238	32	32	771	3.743	4.315	2.714
s641	54	42	810	4.622	4.998	2.400
term1	34	10	1039	8.250	7.072	1.425

Table 1: BDD size and performance measures

Table 1 summarizes the results for the BDDs. For each benchmark we report the number of inputs and outputs, followed by the number of internal nodes in the BDD after reordering. The delays for both minimum sized (*DT*) and double sized (*DT2*) pass transistor circuits, and the delays for the standard-cell implementation are reported.

It appears that the node count is a good measurement for the complexity of the BDD: for a node count up to at least 500 the direct mapping of BDDs to layout seems feasible. Generally functional blocks of this complexity would be mapped in several standard cells. With our approach a single macro-cell will be generated. For higher node counts decomposition into smaller blocks is required for optimal results.

When using double-sized pass transistors the results are sometimes better, but not always. This indicates that the optimum size for the pass transistors is not always the minimal size, but should be chosen according to the performance/area trade-off.

Table 2 shows data on ZBDD mapping. The columns report the number of literals in the flattened and minimized two-level implementation of each benchmark, the number of ZBDD nodes

Bench	Literals	Nodes	Buffers	Delay (ns)	
				ZBDD	Std
b1	11	12	0	0.386	0.487
daio	21	20	2	0.434	0.811
cm150a	81	47	0	0.625	1.049
s208	172	103	10	0.964	1.426
b9	439	194	26	1.047	1.187
s832	887	388	53	1.385	2.374
term1	1953	427	65	2.459	1.425
s641	8277	827	150	1.877	2.400
s1238	12020	1207	185	1.878	2.714
alu4	18347	1784	320	3.077	3.243

Table 2: ZBDD size and performance measures

and the number of buffers inserted because of either excessively long chain of transistors or sneak path elimination. The next two columns show the delays for the ZBDDs and the corresponding delays for the standard-cell implementation.

Overall we can conclude that the direct mapping of ZBDDs to domino style logic gates produces good results.

Bench	BDD-dt	BDD-dt2	Std	ZBDD
alu4	30.482	36.623	23.616	0.490
b1	0.527	0.527	0.423	0.252
b9	3.202	3.371	2.061	1.202
cm150a	0.419	0.466	0.843	0.106
daio	0.625	0.637	0.545	0.377
s208	1.424	1.581	1.434	0.145
s641	28.992	33.559	6.720	2.706
s832	7.483	9.737	4.575	0.246
s1238	30.866	34.924	18.947	1.574
term1	83.423	85.116	2.681	0.350

Table 3: Average power (in mW at 100MHz)

Data on average power consumption (on the same patterns used for timing) is reported in Table 3. Here, the results are less predictable.

5 Conclusions and future work

In this paper we presented some preliminary results on the applicability of direct BDD mapping to pass transistor networks or domino circuits. Such direct mapping procedure is the core of a new synthesis paradigm that tackles the complexity and the uncertainties caused by excessive wiring by adopting a coarse-grain "virtual library" whose cells are automatically generated by fast BDD mapping of functions with many inputs and outputs.

From the results obtained so far, it is quite clear that BDD size is the most important factor in determining the quality of the final implementation. Optimizations such as buffering have beneficial effect for implementations that are not competitive with standard-cell alternatives.

There are many directions of improvement to be investigated. First we need a better buffer insertion algorithms to improve the performance without excessive overhead. Second, automatic transistor sizing for performance enhancement could be explored. Third, when the layout mapping tool for the BDD- and ZBDD-based net lists is available, a more careful analysis of the performance after placement and routing can be done. We expect that the area comparison will be favorable for the BDD and ZBDD mappings, especially for deep-submicron technologies where most of the area is spent on wires.

Acknowledgments

This research was sponsored in part by ARPA, under grant No. DABT63-95-C-0049. We would like to thank Fabio Somenzi for providing the CUDD package used for ZBDD mapping.

References

[1] K. Yano, Y. Sasaki et al., "Top-down pass-transistor logic design," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 6, pp. 792–803, 1996.

[2] M. Tachibana, "Heuristic algorithms for FBDD node minimization with application to Pass-Transistor-Logic and DCVS synthesis," *Workshop on Synthesis and System Integration of Mixed Technologies*, pp. 96–101, 1996.

[3] K. Konishiki, S. Kishimoto et al., "A logic synthesis system for the pass-transistor logic SPL," *Workshop on Synthesis and System Integration of Mixed Technologies*, pp. 32–39, 1996.

[4] K. S. Brace, R. L. Rudell and R. E. Bryant, "Efficient Implementation of a BDD Package," *Design Automation Conference*, pp. 40–45, 1990.

[5] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *International Conference on Computer-Aided Desing*, pp. 42–47, 1993.

[6] S. Minato, "Zero-suppressed BDDs for set manipulation in combinational problems," *Design Automation Conference*, pp. 272–277, 1993.

[7] S. Minato, "Fast weak-division method for implicit cube set representation," *IEEE Transactions on CAD/ICAS*, vol. 15, n. 4, pp. 377–384, 1996.

[8] N. Weste and K. Eshraghian, *Principles of CMOS VLSI design. A systems perspective*, Addison Wesley, 1992.

[9] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Transactions on CAD/ICAS*, vol. CAD-6, n. 5, pp. 727-750, 1987.

[10] V. Bertacco and M. Damiani, "Boolean function representation based on disjoint-support decompositions," *International Conference on Computer Design*, pp. 27–32, 1996.

[11] F. Somenzi. *The CUDD package User's guide. Version 1.0.5* November 1995.