Preprint from Proceedings of IWLS99, Granlibakken, USA, May 1999

On the Generation of Multiplexer Circuits for Pass Transistor Logic

Christoph Scholl Bernd Becker

Institute of Computer Science
Albert–Ludwigs–University
D 79110 Freiburg im Breisgau, Germany
email: <name>@informatik.uni-freiburg.de

Abstract

Pass Transistor Logic has attracted more and more interest during last years, since it has proved to be an attractive alternative to static CMOS designs with respect to area, performance and power consumption. Existing automatic PTL synthesis tools use a direct mapping of (decomposed) BDDs to pass transistors. Thereby, structural properties of BDDs like the ordering restriction and the fact that the select signals of the multiplexers (corresponding to BDD nodes) directly depend on input variables, are imposed on PTL circuits although they are not necessary for PTL synthesis.

General Multiplexer Circuits can be used instead and should provide a much higher potential for optimization compared to a pure BDD approach. Nevertheless – to the best of our knowledge – an optimization of general Multiplexer Circuits (MCs) for PTL synthesis was not tried so far due to a lack of suitable optimization approaches. In this paper we present such an algorithm which is based on efficient BDD optimization techniques. Our experiments prove that there is indeed a high optimization potential by the use of general MCs – both concerning area and depth of the resulting PTL networks.

1 Introduction

Pass Transistor Logic (PTL) has proved to be an attractive alternative to static CMOS designs with respect to area, performance and power consumption [23, 14, 9, 12]. In earlier works using PTL the main disadvantage was that the PTL circuits were designed by hand and there was a lack of automatic synthesis tools.

Recently, several approaches for an *automatic* PTL synthesis flow were proposed [22, 6, 3, 10, 8, 13]. They are all based on a mapping of BDDs [5] (in most cases of decomposed BDDs) to PTL. The advantage of this method is that the PTL circuits originating from BDDs are sneak-path-free [3, 6], i.e. there is no assignment to the inputs which produces a conducting path from power supply to ground. However, BDDs use an ordering restriction, which is not necessary for PTL synthesis. Moreover even the restriction to free BDDs [2] or general BDDs [1] is not necessary. It is easy to see that we can also use general Multiplexer Circuits (MCs)[1] as a basis to synthesize PTL circuits without losing the property of sneak-path absence. Of course, there are more degrees of freedom for MC optimization compared to BDD optimization, since BDDs can be viewed as special cases of MCs. Thus, MCs should provide better PTL solutions than BDDs.

However – to the best of our knowledge – all existing automatic PTL synthesis procedures are based on BDDs. One reason for this could be the fact, that there are efficient BDD packages (see e.g. [19]), which provide efficient BDD optimization techniques by variable reordering like sifting [15], whereas powerful optimization techniques for MCs have been missing. In this paper we present such a powerful optimization procedure for MCs, which makes use of the additional degrees of freedom compared to BDDs. Our novel technique is able to improve on both size and depth of BDD based circuits (see Section 5). Although the result of our algorithm are MCs, we can make use of well matured and efficient BDD optimization techniques to compute the MCs.

In Section 2 we give a comparison between BDDs and MCs. Section 3 reviews how BDDs or MCs are mapped to Path Transistor Logic. In Section 4 we present our algorithm for MC minimization. After giving experimental results for PTL synthesis using this algorithm in Section 5 we conclude the paper with Section 6.

[1] MCs are basically the same as if-then-else DAGs [11].

2 BDDs versus MCs

BDDs provide a canonical representation of Boolean functions. As defined in [5], they are ordered, i.e. on each path from their root to a terminal node each input variable occurs only once and on each path the input variables occur in the same order.

In contrast, Multiplexer Circuits (MCs) are more general:

Definition 1 *A Multiplexer Circuit (MC) M is modeled as a directed acyclic graph (V, E). The node set V is partitioned into four sets V_{const}, V_{inp}, V_{inv} and V_{mux}:*

- *The nodes of V_{const} are constants, have indegree 0 and are labeled by 0 or 1.*
- *The nodes of V_{inp} are inputs, have indegree 0 and are labeled by Boolean variables.*
- *The nodes of V_{inv} are inverters and have indegree 1.*
- *The nodes of V_{mux} are multiplexers and have indegree 3.*

There is a bijective mapping $IN : \{1, \ldots, |V_{inp}|\} \rightarrow V_{inp}$ such that $IN(i)$ defines the ith input of the function defined by the MC M. There is a mapping $OUT : \{1, \ldots, m\} \rightarrow V$ such that $OUT(i)$ defines the ith output of the function defined by the MC M.

Thus MCs are Boolean circuits consisting only of multiplexers, inverters and constants and it is straightforward to define the Boolean function represented by an MC.

Since a BDD node labeled by a variable x_i can be viewed as a multiplexer with select input x_i, it is clear, that BDDs can be viewed as a restricted class of MCs. Because BDDs correspond only to a *restricted* class of MCs, it is also clear, that there are more degrees of

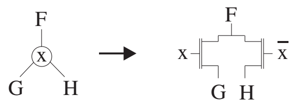

Figure 1: Implementation of a multiplexer by path transistors

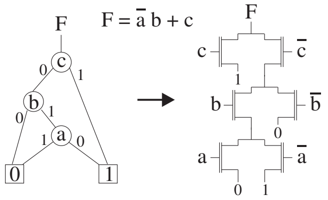

Figure 2: Mapping of a BDD to an NMOS PTL implementation

freedom in MC optimization compared to BDD optimization. However the question arises how to exploit these additional degrees of freedom. Our answer to this question can be found in Section 4.

Before we deal with our approach to MC optimization, we give a brief review of Pass Transistor Logic (PTL) in the next section.

3 Path Transistor Logic

Path Transistor Logic has proved to be an attractive alternative to static CMOS designs[2] with respect to area, performance and power consumption [23, 14, 9, 12, 22, 6, 3, 10, 8, 13].

The basic unit in PTL is a MOS transistor which is used as a switch. It is very easy to implement a multiplexer as a wired OR of two MOS transistors (see Figure 1). For this reason recent automatic PTL synthesis tools use BDDs as a basis for PTL synthesis. Figure 2 shows an example of a BDD mapped to an NMOS PTL implementation. Mapping BDDs to PTL is easy and has the additional advantage that the resulting circuits are sneak-path-free. But note that the same is also true for general Multiplexer Circuits.

4 Our Algorithm for MC minimization

A mapping to PTL is not only easy for BDDs, but also for general MCs. Since MCs are more general, there is a higher potential for optimization both concerning area and depth.

A BDD realizing an n–input Boolean function typically contains paths of BDD nodes/multiplexers of length n, such that the delay of a corresponding PTL implementation is linear in n. More precisely, a chain of n transistors in series even has a quadratic delay in n [21] and buffers have to be inserted after a constant number of levels to achieve a linear delay. We will show in the following that a path of length n can be avoided by using MCs.

To present our algorithm for MC minimization we need the following definition which characterizes special nodes at the bottom of a BDD:

Definition 2 *A* BDD *node is called a* positive variable node *iff its low son is constant* 0 *and its high son is constant* 1*. It is called a* negative variable node *iff its low son is constant* 1 *and its high son is constant* 0 *and it is called a* variable node *iff it is a positive or a negative variable node.*

A BDD *node is called a* multiplexer node *iff both, low son and high son, are a constant node or a variable node and at least one of the sons is a variable node. If both sons of a multiplexer node are variable nodes it is called a* true *multiplexer node, otherwise a* pseudo *multiplexer node.*

Intuitively, our algorithm now successively removes multiplexer nodes from the original BDD thereby replacing "parts of the BDD" by "new" variables. The "meaning" of the new variables is computed in a separate MC. Finally, the whole BDD has been transformed into an MC.

Our algorithm starts with a BDD for a single-output Boolean function. (Note that it can easily be extended to multi-rooted BDDs and BDDs with complemented edges [4].) The algorithm uses a mapping mc_map between $\{x_1, \ldots, x_n\}$ and the input nodes of the MC, i.e., $mc\_map(x_i)$ gives the MC input node labeled by x_i. In the course of the algorithm mc_map is extended to newly introduced variables x, here $mc\_map(x)$ gives the signal line in the MC corresponding to x.

The algorithm now proceeds as follows (for illustration see also Figure 3):

Input: BDD B representing function $f : \{0,1\}^n \rightarrow \{0,1\}$ with input variables $x_1, \ldots, x_n$.

Output: MC for f.

1. (a) Compute all multiplexer nodes of BDD B.
   (b) If there is a true multiplexer node, choose v_{mux} as the true multiplexer node with most incoming edges[3]. If there are only pseudo multiplexer nodes, choose v_{mux} as the pseudo multiplexer node with most incoming edges.
   (c) Build the BDD BDD_c for a new intermediate variable c.
   (d) Replace v_{mux} and the corresponding sub-BDD in B by BDD_c.
   (e) A new multiplexer is introduced in the MC. If v_{mux} is labeled by variable x, the select input of the multiplexer is connected to MC node $mc\_map(x)$. If the low son of v_{mux} is constant 0 (1), the 0-data-input of the multiplexer is connected to constant 0 (1) node of the MC. If the low son is the positive variable y, the 0-data-input of the multiplexer is connected to $mc\_map(y)$ and if the low son is the negative variable $\overline{y}$, the 0-data-input of the multiplexer is connected to a new inverter, which itself is connected to $mc\_map(y)$. The 1-data-input is assigned in the same way.
2. Optimize the resulting BDD B by variable reordering.
3. Repeat steps 1 and 2 until the BDD consists only of one variable node.

Note that reordering can cause a change of the variable label of the next multiplexer node to be replaced. (Experiments using our algorithm for MC optimization show that this happens indeed.)

In each step of the algorithm the initial Boolean function f is represented by two parts: a BDD part and a MC part. Of course, we may interpret the BDD part as an MC. If we connect the select-inputs of the multiplexers for BDD nodes labeled by variable x to $mc\_map(x)$, then we obtain an MC for f.

The MC size achieved so far can be determined by the size of the already constructed MC part and the size of the remaining BDD.

[2] which are in fact restricted cases of PTL

[3] The intuition behind this selection is that this multiplexer node is the "most important" for the computation of the BDD in some sense.

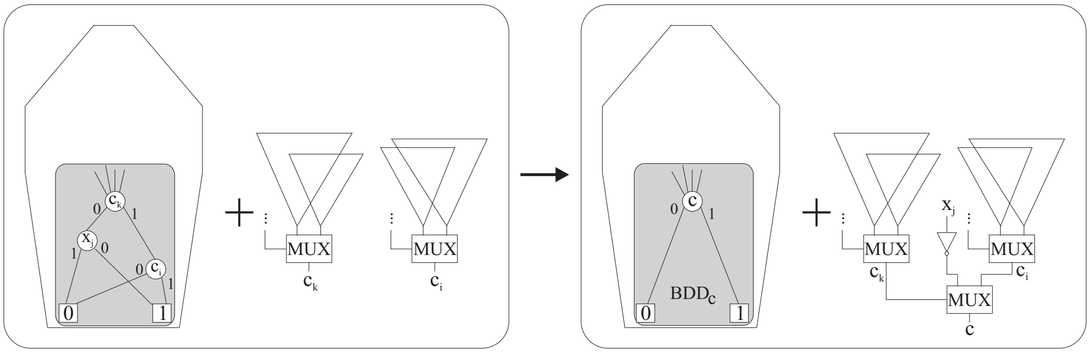

Figure 3: Illustration of step 1) of the algorithm: x_j is a primary input variable, c_i and c_k intermediate variables for multiplexers introduced in previous steps of the algorithm. The multiplexer node labeled by c_k is replaced by a new variable c and a new multiplexer is introduced in the MC which computes the assignment of c. After reordering in step 2) the next selected multiplexer node is not necessarily labeled by c_k.

Optimizing the size of the remaining BDD corresponds to optimizing this preliminary size.

But we can also optimize the *depth* of the current MC circuit: Each variable of the BDD corresponds to a primary input variable or a multiplexer of the already constructed MC. This means that a circuit depth information can be assigned to each BDD variable. If we interpret the BDD part as an MC again, we can compute the current depth of the circuit. Changing the variable order of the BDD does also change the depth of the circuit.

To optimize size and depth of the resulting MC (step 2. of the algorithm) we use a variant of sifting [15], which we call *delay sifting*. (Ordinary) sifting is based on finding the locally optimal position of a variable assuming that all other variables remain fixed. To determine the optimal position of a variable in the variable order it is sifted to all possible positions and then, the position, where the resulting BDD size is minimized, is selected. The cost function during sifting is only the size of the resulting BDD. To take account of our two optimization goals (area and depth) we change the cost function of sifting: We use some combination of BDD size and depth of the overall circuit.

For each position of the variable we determine the new size $size^{new}$ of the resulting BDD and the new depth of the overall circuit $depth^{new}$. Then we choose the position for the variable where the expression

$$\alpha \cdot \frac{size^{new}}{size^{old}} + (1-\alpha) \cdot \frac{depth^{new}}{depth^{old}} \qquad (1)$$

is minimized. ($size^{old}$ and $depth^{old}$, respectively, mean the BDD size and depth of the overall circuit before moving the variable, α is a number between 0 and 1 to influence the trade off between BDD sizes and depth.)

If the already constructed part of the MC circuit gives depth information d_x for variable x at level i, we say that x provides depth contribution $d_x + i$. The depth of the overall circuit is estimated by the maximum depth contribution over all levels i. This gives us only an approximation of the total depth, but the approach has the advantage that the depth estimation can adjusted locally during level exchange, such that asymptotic complexity of delay sifting remains the same as for original sifting.

Figure 4 gives an interesting example for our algorithm to optimize MCs. We consider the $exor$ function with 8 inputs. Note that for this example in each step of the algorithm the function represented by the remaining BDD is totally symmetric, such that changing the position of a variable does not change the BDD size, i.e. in formula 1 only the second part concerning depth plays any role. Starting from a BDD with linear depth our algorithm constructs step by step a MC for the same function. The resulting MC has logarithmic depth. The improvement on the depth is due to the fact that intermediate variables are used as select inputs of multiplexers in our approach.

5 Experimental Results

In this section we present our results for PTL synthesis using the MC optimization algorithm of Section 4. For our experiments we use the implementation of [6] which is integrated in the sis environment [17]. Buch et al. [6] transform a Boolean circuit into a so-called "decomposed BDD" to prevent a size explosion of a monolithic BDD approach. BDDs are constructed starting from the inputs. When a certain size or depth limit of the resulting BDD would be reached, an intermediate variable or cut point is introduced. The result is a set of clusters of the circuit, which are represented by BDDs depending on primary input variables or intermediate cut point variables. After that in [6] the BDDs for these clusters are mapped to PTL. A "PTL cell" is computed for each cluster. To cope with the quadratic delay of transistors in series buffers are inserted for the outputs of the PTL cells.

In this paper we replace the BDD based PTL mapping of [6] by an MC based mapping as described in Section 4. Of course our MC optimization approach can also be used as a post-processing step of other BDD based PTL synthesis tools like [10, 8] to optimize the PTL cells originating from BDD representations.

Since the clusters produced by [6] are very small (the depth of the BDDs is not larger than 3), we first enlarge the clusters to some extent to increase the optimization potential of the MC approach. We remove cut point variables by composition as long as the overall BDD size will not increase in this way. The orders of the BDDs for each cluster are optimized separately. Afterwards clusters with similar support set are transformed into the same variable order again, if this transformation improves the overall BDD size[4]. Moreover all operations are carried out only if a maximum BDD size for a cluster is not exceeded (in our experiments we use a limit of 100). Since the sizes of PTL cells are larger now, we cannot do without buffer insertion within PTL cells to prevent too many unbuffered transistors in series. We insert a buffer (more precisely two inverters), when the longest chain of pass transistors exceeds a given limit (3 in our experiments). After that we perform a greedy inverter minimization similar to [24, 20].

[4] clusters with same variable order can share BDD nodes

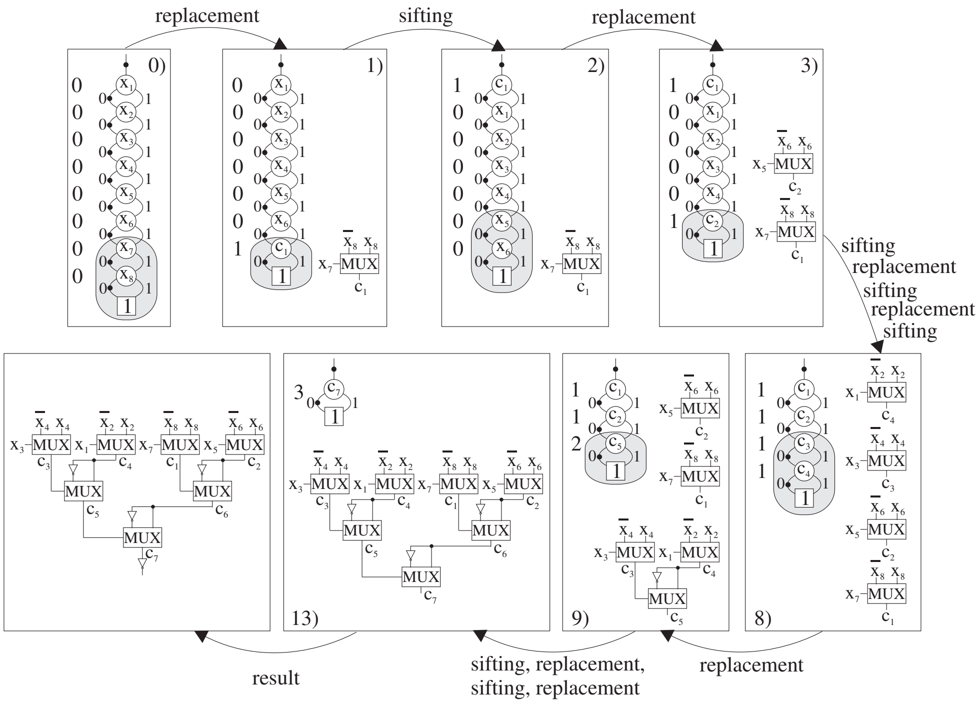

Figure 4: Computation of an MC for $exor(x_1, \ldots, x_8)$. Here we use BDDs with complement edges. The numbers beside the BDD nodes represent the depth informations which are assigned to the nodes and which are used by delay sifting.

We tried two different optimization strategies: optimization only for area (weight $\alpha = 1$, see Section 4) and optimization for a combination of area and depth with $\alpha = 0.3$. Our depth minimization makes use of depth information assigned to the already constructed MC part as described in Section 4. As already proposed in [6], we can additionally use also depth informations for the inputs of the cluster, which is presently optimized, since the clusters which compute these input signals are optimized before.

Table 1 shows our preliminary results for ISCAS89 benchmarks compared to the initial solution of the tool from [6]. Columns 2–4 show the results of the tool from [6], columns 5–8 the results of the area minimization and columns 9–12 the results of the combined area and depth minimization. Columns "mux/inv" give the numbers of multiplexers and inverters of the result, columns "area" give the active transistor area for a realization using only NMOS transistors (the size of an NMOS transistor is assumed to be $1.5\lambda \times 1\lambda$) and columns "md" give the maximum number of multiplexers on a path from primary inputs to primary outputs. Both [6] and our tool use buffer insertion to force the maximum number of transistors in series to be 3. The experiments were performed on a SPARC Ultra 2. Columns "time" give CPU times in seconds to transform the BDDs into MCs both for the area minimization and the combined area and depth minimization.

The experiments prove that there is indeed a high optimization potential of MC minimization compared to BDD minimization:

Our area minimization is able to achieve considerable improvements on the multiplexer/inverter counts and thus also for transistor area in comparison to [6]. In all cases the transistor area is improved (up to 44.49% for C5315). On the average the multiplexer and inverter counts are improved by 28.5% and 30.7% respectively and the transistor area is improved by 29.8%. Interestingly already the area optimization is able to improve the depths of the PTL circuits in 9 out of 11 cases. The overall improvement is 12.8%.

As expected, the combined area and delay optimization needs slightly more area than our results for pure area minimization, but is still better than the results of [6]. (It remains an average area improvement of 23.0%.) The experiments show that we can really exploit an area/depth trade off by our parameter for delay sifting. In all cases the depth results of [6] are improved (up to 39.29% for C2670) while maintaining better area results. On the average the depth results of the area optimization are further improved by 23.4%, such that compared to [6] the depth could be improved by 33.2%.

As already mentioned, an inspection of the resulting MC circuits of our optimization algorithm shows, that they are substantially different from BDD realizations, since we get rid both of the ordering restriction and the restriction to MCs with only input variables as selector inputs of the multiplexers. Thus, we really obtained a general MC structure by using algorithms working on the (restricted) BDD structures.

6 Conclusions and Future Work

In this paper we presented for the first time an automatic PTL synthesis approach which is based on general Multiplexer Circuits rather than on BDDs. Our experiments show, that we are able to exploit the additional degrees of freedom both for area and delay optimization. These degrees of freedom arise from removing restrictions of BDDs, which are important for verification applications, but not for PTL synthesis.

We put our experiments on top of the results of [6], but it is obvious, that our MC optimization approach can also be used as a post-processing step of other BDD based PTL synthesis tools like [10, 8] to optimize the PTL cells originating from BDD representations.

circuit	Berkeley mux/inv	area	md	Our results: area mux/inv	area	md	time	Our results: depth mux/inv	area	md	time
C17	7/12	75	4	6/6	45	3	0.32	6/6	45	3	0.33
C432	207/250	1746	47	196/231	1627.5	51	24.26	214/244	1740	32	20.15
C499	414/413	3100.5	26	302/288	2202	23	40.54	339/337	2533.5	22	39.38
C880	354/401	2866.5	32	313/311	2338.5	29	47.53	323/345	2521.5	21	35.51
C1355	510/465	3622.5	34	326/318	2409	28	44.32	352/326	2523	22	42.83
C1908	416/430	3183	39	307/324	2379	29	43.00	340/367	2671.5	27	30.77
C2670	768/917	6430.5	28	505/506	3792	41	68.80	528/580	4194	17	69.15
C3540	1112/1173	8614.5	52	950/943	7093.5	42	143.16	1017/1040	7731	37	106.14
C5315	1912/2162	15465	47	1183/1119	8584.5	34	176.78	1151/1192	8772	31	152.70
C6288	2698/2764	20532	159	2208/2408	17460	133	276.68	2574/2761	20146.5	100	242.20
C7552	2706/2776	20610	38	1646/1698	12579	28	229.75	1700/1867	13501.5	26	194.17
$\sum$	11104/11763	86245.5	506	7942/8152	60510	441		8544/9065	66379.5	338	

Table 1: Comparison for PTL synthesis

As a future work we plan to incorporate don't care conditions into our approach. Don't cares can be used to minimize the BDD part during the MC computation using methods from [7, 18, 16]. There are two types of don't care informations during MC computation for a cluster of the circuit: satisfiability and observability don't cares which originate from the environment of the cluster and don't cares which originate from the MC part of the cluster that is already computed.

References

[1] P. Ashar, A. Ghosh, and S. Devadas. Boolean satisfiability and equivalence checking using general binary decision diagrams. In *Int'l Conf. on CAD*, 1991.

[2] J. Bern, J. Gergov, C. Meinel, and A. Slobodová. Boolean manipulation with free BDD's. First experimental results. In *European Design & Test Conf.*, pages 200–207, 1994.

[3] V. Bertacco, S. Minato, P. Verplaetse, L. Benini, and G. De Micheli. Decision diagrams and pass transistor logic synthesis. In *Int'l Workshop on Logic Synth.*, 1997.

[4] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[5] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[6] P. Buch, A. Narayan, A.R. Newton, and A.L. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *Int'l Conf. on CAD*, pages 663–670, 1997.

[7] S. Chang, D. Cheng, and M. Marek-Sadowska. Minimizing ROBDD size of incompletely specified multiple output functions. In *European Design & Test Conf.*, pages 620–624, 1994.

[8] R. Chaundry, T.-H. Liu, A. Aziz, and J.L. Burns. Area-oriented synthesis for pass-transistor logic. In *Int'l Workshop on Logic Synth.*, pages 169–176, 1998.

[9] T.S. Cheung and K. Asada. Regenerative pass-transistor logic: A circuit technique for high speed digital design. *IEICE Trans. Electron.*, E79-C(9):1274–1283, 1996.

[10] F. Ferrandi, A. Macii, E. Macii, M. Poncino, R. Scarsi, and F. Somenzi. Symbolic algorithms for layout-oriented synthesis of pass transistor logic circuits. In *Int'l Conf. on CAD*, 1998.

[11] K. Karplus. ITEM: an if-then-else minimizer for logic synthesis. Technical report, University of California, Santa Cruz, 1992.

[12] F.S. Lai and W. Hwang. Design and implementation of differential cascode voltage switch with pass-gate (dcvspg) logic for high-performance digital systems. *IEEE Jour. of Solid-State Circ.*, 32(4):563–573, April 1997.

[13] T.-H. Liu, A. Aziz, and J.L. Burns. Performance driven synthesis for pass-transistor logic. In *Int'l Workshop on Logic Synth.*, pages 255–259, 1998.

[14] A. Parameswar, H. Hara, and T. Sakurai. A high speed, low power, swing restored pass-transistor logic based multiply and accumulate circuit for multimedia applications. In *Proc. Custom Integrated Circuits Conf.*, pages 278–281, May 1994.

[15] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[16] C. Scholl, S. Melchior, G. Hotz, and P. Molitor. Minimizing ROBDD sizes of incompletely specified functions by exploiting strong symmetries. In *European Design & Test Conf.*, pages 229–234, 1997.

[17] E. Sentovich, K. Singh, L. Lavagno, Ch. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[18] T.R. Shiple, R. Hojati, A.L. Sangiovanni-Vincentelli, and R.K. Brayton. Heuristic minimization of BDDs using don't cares. In *Design Automation Conf.*, pages 225–231, 1994.

[19] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.

[20] A. Wang. *Algorithms for Multilevel Logic Optimization*. PhD thesis, U. C. Berkeley, April 1989.

[21] N. Weste and K. Eshraghian. *Principles of CMOS VLSI Design: A Systems Perspective*. Addison-Wesley, 1992.

[22] K. Yano, Y. Sasaki, K. Rikino, and K. Seki. Top-down pass-transistor logic design. *IEEE Jour. of Solid-State Circ.*, 31(6):792–803, June 1996.

[23] K. Yano, T. Yamanaka, T. Nishida, and M. Satio. A 3.8-ns cmos 16 $\times$ 16-b multiplier using complementary pass-transistor logic. *IEEE Jour. of Solid-State Circ.*, 25(2):388–395, April 1990.

[24] H. Zhou and A. Aziz. Buffer minimization in pass transistor logic. In *Int'l Workshop on Logic Synth.*, pages 155–161, 1998.