Generalized Buffering of PTL Logic Stages using Boolean Division

Rajesh Garg Sunil P Khatri
Department of EE, Texas A&M University, College Station, TX 77843

Abstract

Pass Transistor Logic (PTL) is a well known approach for implementing digital circuits. In order to handle larger designs, and also to ensure that the total number of series devices in the resulting circuit is bounded, partitioned Reduced Ordered Binary Decision Diagrams (ROBDDs) can be used to generate the PTL circuit. The output signals of each partitioned block typically needs to be buffered. In this paper, we present a methodology to perform *generalized* buffering of the outputs of PTL blocks. By performing the Boolean division of each PTL block using different gates in a library, we select the gate that results in the largest reduction in the height of the PTL block. In this manner, these gates serve the function of buffering the outputs of the PTL blocks, while also reducing the height and delay of the PTL block. Over a number of examples, we demonstrate that our approach results in a 26% reduction in circuit delay and number of MUXes required, with a modest improvement in circuit area, compared to a traditional buffered PTL implementation of the circuit.

1. Introduction

Pass Transistor Logic (PTL) is a circuit implementation style that has typically been used for many specific circuit implementations, like barrel shifters. Although PTL offers great benefits for such designs, there has been no widely accepted PTL design methodology that could be used to make PTL more broadly acceptable. There have however been several efforts, at the research level, to make this a reality.

Synthesis approaches for PTL structures typically leverage the fact that there is a direct mapping between the ROBDD [1, 2] and the PTL implementation of a circuit. In fact, each ROBDD node can be mapped to a MUX (which can be implemented using NMOS or CMOS devices). Figure 1 illustrates the mapping between an ROBDD node and a MUX. For the PTL implementation of the ROBDD of any function, there is an isomorphism between the connectivity of the ROBDD nodes and the MUXes in the PTL implementation. This elegant and easy mapping between ROBDDs and PTL structures is not without attendant problems, however.

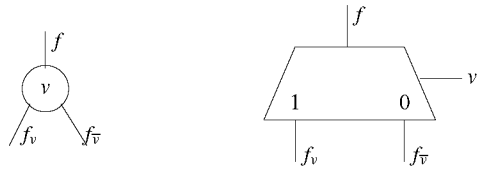


Figure 1: ROBDD Node and its MUX based Implementation

- For one, in a bulk MOS implementation of any circuit, it is not practical to connect more than 4-5 devices in series, due to the phenomenon called *body effect*, which effectively increases the threshold voltage V_T of most of the series-connected MOSFETs. This results in a slower design. Body effect is governed by the equation

  $$V_T = V_T^0 + \gamma\sqrt{V_{sb}}$$

  Here V_T is the threshold voltage of the device, V_T^0 is the threshold voltage of the device at zero body bias, γ is the body effect coefficient, and V_{sb} is the source to bulk voltage of the MOSFET. When several MOSFETs are connected in series in a PTL circuit, all but one of them is affected by body effect. For this reason, in practice, VLSI designers do not stack more than 4-5 devices in series.

  This results in a problem for the traditional PTL implementations, which attempted to build monolithic ROBDDs. To solve this problem, we need to build ROBDDs in a partitioned [3] manner, such that if an intermediate ROBDD has a depth greater than 4 or 5, a new variable is created. In circuit terms, a buffer is implemented at the output of the new variable, ensuring that the circuit drive capability is regenerated every few levels in the final PTL design.

- The second problem is that ROBDDs, if built monolithically, can exhibit unpredictable memory explosion. This precludes the applicability of the PTL methodology for large designs, *as long as the ROBDDs are build monolithically* (since it is very hard to build monolithic ROBDDs of large functions). Again, this suggests the use of partitioned ROBDDs to avert this problem.

In summary, partitioned ROBDD construction helps tackle both the above problems associated with PTL synthesis. In such a partitioned PTL design, the outputs of each PTL structure is buffered, to regenerate the electrical drive capability every 4 or 5 levels.

This work describes a new PTL synthesis approach which performs better than buffered partitioned ROBDD based PTL synthesis. In principle, it still employs partitioned ROBDDs, however, the buffering between PTL stages is done using *generalized buffers*. These could be arbitrary gates in the cell library. We achieve this by casting the problem of generalized buffering as an instance of Boolean division. By using more complex gates for the buffering logic, our method is able to significantly simplify the logic in the PTL structure, resulting in a significant improvement (about 26% on average) in total circuit delay, which is achieved by significantly reducing the total number of MUXes. Even after accounting for the increased area of the generalized buffers, our method comes out slightly ahead (about 2.3% on average). The above comparisons are against traditionally buffered partitioned ROBDD-based PTL structures.

The remainder of this paper is organized as follows: Section 2 discusses some previous work in this area. In Section 3 we describe our method of generalized buffering of PTL structures, based on Boolean division. In Section 4 we present experimental results comparing our idea with traditional buffered partitioned ROBDD-based PTL designs. Conclusions and future work are discussed in Section 5.

2. Previous Work

There has been a significant amount of work in the area of pass transistor logic synthesis. In addition, there have been several reported design variations on the PTL circuit concept. A good reference

source for published work in this area is [4]. The focus of our paper being logic synthesis for PTL design, we will not discuss circuit-level variations of the PTL approach. Given the generality of our synthesis methodology, it is orthogonal to the circuit issues and ideas that are discussed in the papers referred to in [4].

In [5], the authors present a synthesis tool, which works with their PTL library of cells, macrocells etc. The philosophy of [6] is that while PTL based synthesis is well researched, layout and back-end tools for PTL are not found as readily. They describe a layout generator, to help develop more complete tools for PTL design. The approach of [7] is motivated by the need to develop high density, low power, high performance circuits using PTL. The authors report a synthesis method, using two approaches – a modified Karnaugh map [8] approach for small circuits, and a Quine-McCluskey [9, 10] based approach for larger designs. In [11], the authors study regenerative pass transistor logic (RPL), a dual rail PTL family. Their interest stems from the compact area of these circuits. In [12], the authors present a layout and logic synthesis approach, with the input being a logic function, while the output is a PTL netlist, using MUXes and inverters. After synthesizing the logic, the methodology adds inverters, in order to ensure that the longest series path of MUXes has a bounded depth. In this approach, the buffering is simple (using inverters) as opposed to the generalized buffering (using arbitrary library gates) in our work. The work of [13] builds ROBDDs [1, 2] in a partitioned manner [3], thereby avoiding the memory blow-up that often occurs while using ROBDDs. The resulting small ROBDDs are directly mapped to PTL structures, resulting in an efficient synthesis methodology. The downside of this approach is once again, the absence of generalized buffering. The authors allude to performing PTL synthesis in a manner that maps some nodes to CMOS gates while others are mapped into PTL blocks. However, this leaves the designer little control of the depth of these sections. In contrast, our generalized buffering approach guarantees a fixed bound on the depth of the PTL block, and also ensures just one level of generalized buffers between PTL blocks.

In [14], an approach using multilevel gates along with PTL and transmission gates is reported, producing a regular and dense layout. This approach permits buffer inclusion. It can be viewed as an approach that combines synthesis and layout in the PTL design flow, and is as such orthogonal to our approach. The work of [15] reports synthesis algorithms for PTL. Minimization is performed by using incomplete transmission gates. The results of synthesis using their tool PAVOS demonstrate good quality results when compared with manually tuned PTL circuits. The work of [16] reports on PTL synthesis for Complementary Pass Transistor Logic (CPL), Dual Pass Transistor Logic (DPL) and Dual Voltage Logic (DVL). Finally, the work of [17] reports on a novel method to minimize power in a PTL structure, by transforming the problem into one of ROBDD decomposition and solving it with a max-flow min-cut approach.

In [18], a mixed PTL-CMOS approach was presented. The decomposition process simply extracted unate variables from a sum-of-products (SOP) representation of a function and recursively cofactored these variables (creating MUXes in the process) until unate leaves were reached. Unate leaves were realized using library gates alone. The weakness of this approach is that it starts with a SOP representation, limiting its effectiveness for large designs. Further, if a function is unate, then the entire circuit realization is purely standard-cell based. Our approach, in contrast, can handle arbitrarily large designs since it starts with a partitioned ROBDD construction as the first step. Also, it works equally effectively for unate designs.

Another mixed PTL-CMOS approach was presented in [19]. In this paper, the circuit structure was a AND gate followed by a MUX. In contrast to our approach, this scheme did not selectively divide a library of standard cells into the PTL structure, thereby not exploiting the flexibility available in the Boolean division process.

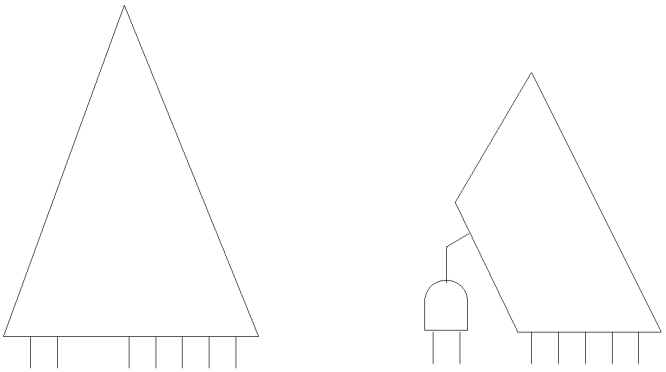
Our approach of performing partitioned ROBDD construction, with generalized buffers (library gates) doing the task of buffering the outputs of PTL structures (and also performing computation) is a novel contribution in this space. It is orthogonal to much of the work on new PTL circuit design ideas, and yields impressive improvements (about 25% on average) in terms of circuit speed, with a nominal area improvement (about 2.5% on average) compared to the traditional buffering that is performed in a partitioned ROBDD based PTL design approach.

3. Our Approach

In order to implement generalized buffering, we employ Boolean division. Consider a Boolean network η, which is initially decomposed using 2-input gates and inverters only. We require that the ROBDDs (which will eventually be utilized to construct the PTL circuit) have a depth of no more than 5 variables.

In our approach, we first build ROBDDs of the nodes of η, topologically from the inputs to the outputs. When we encounter a new node n for which we want to construct a ROBDD, the ROBDD of each of its fanins must have a depth of at most 4 ROBDD variables. When constructing the ROBDD of n, it can initially have at most 8 ROBDD variables.

The key idea is that we will take the ROBDD of n, and attempt to divide it with the gates in a library. If such a Boolean division is possible *and* it is strictly height-reducing, we select it. The test for whether a library gate g, with an associated variable G, divides the ROBDD f of n is described next. A pictorial view of the process is shown in Figure 2.



a) Before dividing AND gate

b) After dividing AND gate (note height reduced otherwise the gate is rejected

Figure 2: Dividing a Generalized Buffer into a PTL Structure

3.1 Boolean Division with Library Gates

DEFINITION 1. *g is a* **Boolean divisor** *of f if h and r exist such that $f = gh + r$, $gh \neq \emptyset$.*

g is said to be a **Boolean factor** *of f if, in addition, $r = \emptyset$, i.e., $f = gh$.*

In this case, h is called the **quotient** and r is called the **remainder**. Note that h and r are not unique.

THEOREM 3.1. *If $fg \neq \emptyset$, then g is a Boolean divisor of f.*

PROOF. If $fg \neq \emptyset$, we can write
$f = fg + f\overline{g}$
$f = g(f + x) + f\overline{g}$ where $x \subseteq \overline{g}$
which is of the form $f = gh + r$. □

However, if f is an Incompletely Specified Function (ISF), with don't care d, then

$f = G(f+d+\overline{g})+(f+d)\overline{g}$.

Therefore, the upper and lower bounds (U and L) for f are:

$U = G(f+d+\overline{g})+(f+d)\overline{g}$

and

$L = (fG+f\overline{g})$

As a consequence, the test that we perform when we want to try to divide a gate g (having a variable G) into the ROBDD f is shown in Algorithm 1. The computations in Algorithm 1 assume that there are no don't cares d.

Algorithm 1 Pseudocode for Division of f by $g \equiv G$

```
test_division(f, g, G){
if fg ≠ 0 then
  L = (fG + f·g')(g ⊕' G)
  U = (fG + G·g' + f·g')(g ⊕' G)
  Z = bdd_between(L, U)
  Z* = bdd_smooth(Z, gvars)
  R = bdd_compose(Z*, G, g)
  if R = f then
    return(success, Z*)
  end if
else
  return fail
end if
}
```

In Algorithm 1, the functions L and U are ANDed with $(g\overline{\oplus}G)$ to express the fact that g and G are not independent variables, but rather are related as $G \equiv g$. We next find a small ROBDD Z (using the function *bdd_between*(), which returns the heuristically smallest ROBDD Z such that $L \subseteq Z \subseteq L+U$), which is a Boolean divisor of f. Next, we smooth out[1] the variables of g. If the resulting ROBDD Z^*, with g composed back into G, is identical to f, we return Z^* as the quotient.

When we perform ROBDD construction, the maximum height of the ROBDD of a node n before division can be 8, as discussed earlier. Our division routines systematically reduce this height to below 5, by using several generalized buffers. If an ROBDD is not able to be divided (and the resulting height after division is greater than 5), then we back-track, and make one of the fanins of n a new variable. The fanin which is made a new variable is the one whose topological level is one less than that of n. The reason for this is illustrated in Figure 3.

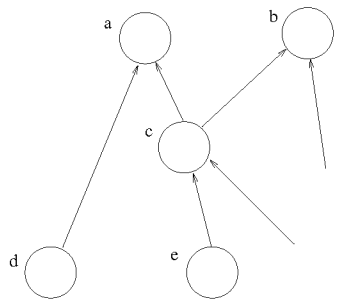


Figure 3: Back-tracking during Division

Consider node a, at topological level n. Suppose it has two fanins c and d, with levels $n-1$ and $n-2$ respectively. Since partitioned ROBDD construction occurs in topological order from inputs to outputs, it is possible that node b has already been processed, and divided. Its ROBDD therefore has a depth less than 5 nodes. Now suppose the ROBDD depths of d and c are 4 each, and the ROBDD of a has depth 8. Suppose all divisions for node a fail. In that case, we need to back-track and make either c or d a new variable. This would guarantee that a has a new depth 5. We select c (with level $n-1$) as the new variable, since it is more likely that d (which is at a lower topological level) has more fanouts at level n or $n-1$. When c is made a new variable, all its fanouts are checked. If any of them has already been processed (such fanouts must have level n), then their ROBDDs are re-computed, and division is re-done. In this way, the PTL network is kept as small as possible. If this re-computation were not performed, then the logic of the node c would be implemented twice as a PTL structure (once for the node b and then again for the variable corresponding to node c itself).

If, after a back-track, the depth of the node n is less than 5, then we continue to grow the corresponding ROBDD further, up to a depth 8.

After attempting division (regardless of whether the division succeeded or failed), the depth of n is guaranteed to be less than or equal to 5, allowing for an elegant exit in case division fails. In general, however, this division strategy yields a number of good generalized buffers, so this occurrence is rare.

4. Experimental Results

We implemented the generalized buffering algorithm in SIS [20]. Our code consisted of reading a circuit (which was decomposed beforehand into 2-input gates and inverters), and then building ROBDDs of its nodes in a topological manner from inputs to outputs. When the height of the ROBDD of any node grew beyond 5, division was invoked, as described in Section 3. The resulting gates that were divided, and the new ROBDD after division were stored within each node's data structures. Since the ROBDDs in question were small (with a maximum height of 8), we performed division exhaustively. Our library consisted of the gates AND2, AND3, AND4, OR2, OR3 and OR4. Since any divided gate becomes a new variable, it is required in both its polarities. Therefore, by DeMorgan's law, we only have non-inverting gates in our library.

Table 1 reports the results we obtained using our algorithm. In this table, column 1 lists the example under consideration. Column 2 lists the number of inverters required by partitioned ROBDD based decomposition of the circuit, using traditional buffering. The traditional method used for comparison was similar to that reported in [13]. The remaining columns report the number of library gates invoked by our algorithm. Note that in general, a healthy number of library gates are utilized for each example.

Table 2 compares the results of traditionally buffered partitioned ROBDD based PTL implementations, with our generalized buffering methodology. Columns 2, 3 and 4 report the circuit delay, area and number of MUXes utilized for the traditional method. Columns 5, 6 and 7 report these numbers for our method (*as a fraction* of the corresponding numbers for the traditional method). Finally, Column 8 reports the runtime for our division based generalized buffering based synthesis algorithm.

In each case, delays were extracted by finding the longest delay path from any output to any input. The MUX structures and all gates in the library were characterized for delay in SPICE [21], using a 100nm BPTM [22] process technology. The area computation was performed by determining the active area of the MUXes and all the library cells. The number of MUXes is simply the sum of the sizes of each of the partitioned ROBDDs in the design.

We observe that our method results in a speed-up of about 26% on average, compared to the traditional method. Also, our method utilizes about 24% fewer MUXes. However, since the generalized buffers occupy greater area than the traditional buffers, the overall area improvement of our method is not as high (2.3% on average).

[1]Smoothing is also referred to as Existential Quantification. The existential quantification of f with respect to a variable x is expressed as $\exists_x f = f_x + f_{\overline{x}}$. Smoothing a set of variables is achieved by performing existential quantification, one variable at a time

These results indicate that generalized buffering offers significantly faster designs, with a small area benefit as well, compared to traditionally buffered PTL. The run-time of our partitioning algorithm is slightly less than 90 seconds for the largest example in our benchmark suite.

The effectiveness of our method can be augmented by invoking dynamic variable re-ordering while performing ROBDD construction. This will allow further reduction in circuit size for both the traditional and generalized buffering methodologies. Generalized buffering can benefit yet further by utilizing multi-level don't-cares, as described in Section 3, something that the traditional method cannot benefit from.

	Traditional	Generalized Buffering						
Ckt	INV	INV	AND2	AND3	AND4	OR2	OR3	OR4
alu2	1994	874	64	20	6	62	22	4
alu4	12098	5940	348	158	82	369	144	132
apex6	3809	2313	118	6	4	112	31	6
C432	1055	943	15	3	0	13	0	0
C499	1332	932	0	16	16	16	0	0
C880	1625	1083	47	9	9	0	0	0
C1908	1883	1445	17	13	1	44	2	0
C3540	6203	3828	154	66	99	160	41	8
C5315	6691	6623	47	11	6	45	4	1
x3	3955	2678	64	12	2	120	26	3
i8	6627	2151	298	117	0	165	29	3
x1	1556	777	49	8	2	27	15	14
pair	7870	5138	201	28	15	185	73	63
rot	2884	1929	76	20	3	49	17	2
C6288	15610	11409	751	25	12	385	41	23
C1355	1995	1792	22	0	0	9	0	0
des	22697	14623	452	182	132	387	69	9
s38417	55283	36862	994	472	56	1765	172	125
seq	11613	5660	396	157	56	358	166	126
too_large	1573	753	48	5	0	42	25	17

Table 1: Number of Library Gates Utilized

	Traditional Buffering			Generalized Buffering			
Ckt	Delay (ps)	Area (μ^2)	#MUX	Delay ratio	Area ratio	#MUX ratio	Time (s)
alu2	5343.66	159.52	709	0.51	0.89	0.602	0.370
alu4	20116.08	967.84	4237	0.40	1.11	0.687	9.180
apex6	1609.02	304.72	1335	0.65	0.96	0.747	1.230
C432	3159.00	84.40	388	0.96	1.01	0.948	0.350
C499	1033.20	106.56	408	1.01	0.92	0.764	0.350
C880	2061.72	130.00	568	0.77	0.84	0.765	0.230
C1908	4092.48	150.64	654	0.73	0.96	0.838	0.920
C3540	21334.86	496.24	2144	0.71	1.08	0.808	4.170
C5315	4967.28	535.28	2347	1.19	1.07	1.02	28.280
x3	1081.98	316.40	1400	0.79	0.96	0.791	1.200
i8	1768.32	530.16	2574	0.61	0.73	0.455	1.560
x1	953.46	124.48	541	0.67	0.93	0.652	0.230
pair	2418.48	629.60	2859	0.62	1.07	0.782	3.300
rot	2058.48	230.72	1006	0.73	0.94	0.782	0.780
C6288	35626.68	1248.80	5487	1.00	1.07	0.832	11.040
C1355	2138.94	159.60	656	1.00	0.96	0.9375	0.660
des	2519.28	1815.76	7803	0.89	0.91	0.787	19.030
s38417	7614.72	4422.64	19872	0.75	0.99	0.795	82.100
seq	7647.84	929.04	3971	0.37	1.13	0.7088	8.340
too_large	3341.34	125.84	554	0.54	1.01	0.641	0.330
average				0.745	0.977	0.767	8.683

Table 2: Comparison of Generalized and Traditional Buffered PTL

5. Conclusions

In this paper, we have presented a Pass Transistor Logic (PTL) circuit synthesis scheme. In order to handle larger designs, and also to ensure that the total number of series devices in the resulting circuit is bounded, partitioned Reduced Ordered Binary Decision Diagrams (ROBDDs) have been used to generate the PTL circuit. Our approach utilizes partitioned ROBDDs to construct the PTL circuit, with the interfaces between these PTL structures buffered using library gates. Our technique of *generalized* buffering requires Boolean division of the PTL block using different gates in a library. In this way, we can select the gate that results in the largest reduction in the height of the PTL block. In this manner, we can have these gates serve the function of buffering the outputs of PTL blocks, and also perform circuit computations at the same time. Over a number of examples, we demonstrate that on average, our approach results in a 26% reduction in delay, and a 2.3% improvement in circuit area, compared to a traditional buffered PTL implementation.

References

[1] R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Trans. Computers*, vol. C-35, pp. 677–691, Aug. 1986.

[2] K. S. Brace, R. L. Rudell, and R. E. Bryant, "Efficient Implementation of a BDD Package," in *Proc. of the Design Automation Conf.*, pp. 40–45, June 1990.

[3] R. K. Brayton, G. D. Hachtel, A. L. Sangiovanni-Vincentelli, *et al.*, "VIS: A system for verification and synthesis," in *International Computer-Aided Verification Conference* (R. Alur and T. A. Henzinger, eds.), vol. 1102, (New Brunswich, NJ), pp. 428–432, Springer-Verlag, July–August 1996.

[4] K. Taki, "A survey for pass-transistor logic technologies-recent researches and developments and future prospects," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 223–226, Feb 1998.

[5] K. Yano, Y. Sasaki, K. Rikino, and K. Seki, "Top-down pass-transistor logic design," *IEEE Journal of Solid-State Circuits*, vol. 31, pp. 792–803, June 1996.

[6] L. Macchiarulo, L. Benini, and E. Macii, "On-the-fly layout generation for ptl macrocells," in *Proceedings, Design, Automation and Test in Europe Conference*, pp. 546–551, March 2001.

[7] D. Radhakrishnan, S. Whitaker, and G. Maki, "Formal design procedures for pass transistor switching circuits," *IEEE Journal of Solid-State Circuits*, vol. 20, pp. 531–536, April 1985.

[8] E. McCluskey, *Logic design principles : with emphasis on testable semicustom circuits*. Prentice-Hall, 1986.

[9] W. Quine, "The problem of simplifying truth functions," *American Math. Monthly*, vol. 59, pp. 521–531, 1952.

[10]

[11] T. Cheung, K. Asada, and H. Wong, "Design automation algorithms for regenerative pass-transistor logic," in *Proceedings of 1997 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, pp. 1540–1543, June 1997.

[12] S.-F. Hsiao, J.-S. Yeh, and D.-Y. Chen, "High-performance multiplexer-based logic synthesis using pass-transistor logic," in *Proceedings of 2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 2, (Geneva), pp. 325–328, May 2000.

[13] P. Buch, A. Narayan, A. Newton, and A. Sangiovanni-Vincetelli, "Logic synthesis for large pass transistor circuits," in *Proceedings, IEEE/ACM International Conference on Computer-Aided Design*, pp. 663–670, Nov 1997.

[14] J. Neves and A. Albicki, "A pass transistor regular structure for implementing multilevel combinational circuits," in *Proceedings, Seventh Annual IEEE International ASIC Conference and Exhibit*, pp. 88–91, Sept 1994.

[15] C. Pedron and A. Stauffer, "Analysis and synthesis of combinational pass transistor circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, pp. 775–786, July 1988.

[16] D. Markovic, B. Nikolic, and V. Oklobdzija, "General method in synthesis of pass-transistor circuits," in *Proceedings. 22nd International Conference on Microelectronics*, vol. 2, pp. 695–698, May 2000.

[17] S. Shelar and S. Sapatnekar, "An efficient algorithm for low power pass transistor logic synthesis," in *Proceedings, Asia and South Pacific Design Automation Conference and the 15th International Conference on VLSI Design*, pp. 87–92, Jan 2002.

[18] Y.-T. Lai, Y.-C. Jiang, and H.-M. Chu, "BDD decomposition for mixed CMOS/PTL logic circuit synthesis," in *Proceedings, IEEE International Symposium on Circuits and Systems*, pp. 5649–5652, May 2005.

[19] S. Yamashita, K. Yano, Y. Sasaki, Y. Akita, H. Chikata, K. Rikino, and K. Seki, "Pass-transistor/CMOS collaborated logic: The best of both worlds," in *Digest of Technical Papers, Symposium on VLSI Circuits*, pp. 31–32, June 1997.

[20] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: A System for Sequential Circuit Synthesis," Tech. Rep. UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[21] L. Nagel, "Spice: A computer program to simulate computer circuits," in *University of California, Berkeley UCB/ERL Memo M520*, May 1995.

[22] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," in *Proc. of IEEE Custom Integrated Circuit Conference*, pp. 201–204, Jun 2000. http://www-device.eecs.berkeley.edu/ ptm.