Maximum Current Estimation in Programmable Logic Arrays*

S. Bobba and I. N. Hajj
Coordinated Science Lab & ECE Dept.
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801
E-mail: {bobba, i-hajj}@uiuc.edu

Abstract

Programmable logic array (PLA) is a circuit realization for the two-level sum of products representation of a multi-output Boolean function. The current drawn by a PLA is input dependent and it makes the problem of estimating the maximum current intractable. Integrated circuit reliability and signal integrity are related to the maximum current drawn by the circuit. Hence, an estimate of the maximum current is required for the design of a reliable VLSI circuit. In this paper, we present an input pattern-independent algorithm to obtain the estimate of maximum and minimum currents drawn by a PLA over all possible input vectors. Experimental results on several benchmark circuits and comparisons with exhaustive simulations are also included in this paper.

1 Introduction

With the demand for high reliability in present day VLSI design, it is essential to estimate the maximum current and the maximum instantaneous energy dissipation for a VLSI circuit early in the design cycle. Large instantaneous energy dissipation causes overheating of devices and degrades circuit performance. Every 10°C increase in the operating temperature roughly doubles the failure rate for the IC [1]. With higher levels of integration, higher clock rates, and shrinking line width, the power bus interconnects are more susceptible to electromigration (EM) induced failure. The Mean-Time-To-Failure (MTTF) of an interconnect due to EM depends on the current density through the interconnect [2]. The minimum and maximum current estimates can be used to determine the severity of these problems early in the design cycle. In this work, our focus is on two-level circuits implemented as a programmable logic array.

Programmable logic array (PLA) is a circuit realization for the two-level sum of products representation of a multi-output Boolean function. A PLA provides a regular structure for implementing combinational and sequential functions. There are several advantages of using PLA for implementing a multi-output Boolean function. The structure of a PLA is regular and it can be generated using an automated process. The programming of the PLA can be modified at later stages of the design process without having to re-do the whole layout. PLA based implementation of complex Boolean functions is an option to the designer to reduce the design time, design effort and meet the cost or time-to-market goals [3].

The current drawn by a PLA is dependent on the applied input vector. This input pattern dependence makes the problem of estimating maximum or minimum current extremely difficult. The number of input vectors that have to be simulated to find the maximum or minimum current drawn by a PLA is exponential in the number of inputs to the PLA.

*This work was supported by SRC under contract 96-DP-109

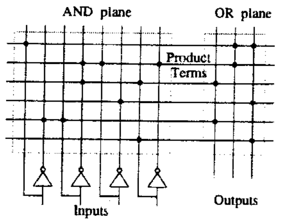

Figure 1: Programmable Logic Array

In [4], Devadas et al. transform the problem of estimating maximum power dissipation in a static or dynamic CMOS circuit (two-level or multi-level circuit) to a weighted max-satisfiability problem on a set of multi-output Boolean functions obtained from the circuit logic description. Then they use either a disjoint cover enumeration algorithm or a branch-and-bound algorithm to solve the weighted max-satisfiability problem, which is an NP-complete problem. The complexity of their algorithm is exponential in the number of primary inputs and the analysis is slow even for small circuits. In this work, we present an input pattern-independent algorithm to obtain the estimate of maximum and minimum current drawn by a PLA over all possible input vectors. The maximum current estimate can be used to find the peak instantaneous energy dissipation. The minimum current is a lower bound on the average current. This can be used to find a lower bound on the width of a power bus interconnect and find interconnects that are guaranteed to be susceptible to EM induced failure.

This paper is organized as follows. In the next section, we formulate the problem. In section 3, we describe different implementation styles and the method to construct the constraint graph. We also present the duality relationship used to find the maximum and minimum currents for a NOR-NOR implementation of the PLA. In section 4, we describe the algorithms used to obtain the maximum and minimum currents drawn by a PLA. In section 5, experimental results and comparisons with exhaustive simulations are presented. Finally, in section 6 we give the conclusions.

2 Problem Formulation

A typical PLA uses an AND-OR structure as shown in Fig. 1. The basis for a PLA is the sum of products representation of Boolean functions. The AND plane takes in the PLA inputs and generates the product terms. Each product term consists of a set of literals that correspond to the PLA inputs. The OR plane takes in the product terms and generates the PLA outputs. Each row of the AND plane evaluates a product term and each column of the OR plane evaluates a PLA output. Each of these functions are usually implemented using a pseudo-NMOS design style. Fig. 2 shows a dynamic pseudo-NMOS circuit.

<u>Definition</u> - **Logic block**: It denotes a pseudo-NMOS circuit implementing a row (column) of the AND (OR) plane.

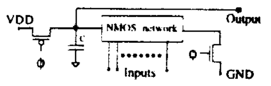

Figure 2: Dynamic pseudo-NMOS circuit

The number of logic blocks in a PLA circuit is equal to the sum of the number of rows in the AND plane and the number of columns in the OR plane. The term output node is used to refer to the output of a logic block. The NMOS network shown in Fig. 2 implements either an AND or OR function of the inputs. A output node can take two logic values, logic high denoted by HI and logic low denoted by LO. OFF-SET (ON-SET) of the output node is defined as the set of input vectors for which the output node is LO (HI). When an input vector is applied, the output node is pulled LO if the NMOS network turns ON. The NMOS network is ON when the input vector applied belongs to the OFF-SET of the output node. In a static pseudo-NMOS circuit, current is drawn from the power bus through the PMOS transistor when the output node is LO. When the output node is HI, no current is drawn from the power bus (ignore leakage current). In a dynamic CMOS circuit, the output node is precharged to HI, and during the evaluate phase the output node is pulled LO if the NMOS network turns ON. In the next precharge phase, all the output nodes that were discharged draw simultaneous charging current. If we assume that all the logic blocks draw the same current, the maximum current drawn from the power bus corresponds to the maximum number of output nodes that are LO. This argument is valid for both the static and dynamic implementations of the logic blocks. To determine an input vector that sets maximum number of output nodes to LO, one has to find an input vector that is in the OFF-SET of maximum number of output nodes. This problem is the max-satisfiability problem. If the current drawn by the logic blocks is different, then the current values can be used as weights and the problem becomes a weighted max-satisfiability problem. The problem of determining the minimum current corresponds to finding an input vector that is in the ON-SET of the maximum number of output nodes. This minimizes the number of nodes that are LO and hence, minimizes the current drawn from the power bus. If a weight is associated with each node then this problem also becomes a weighted max-satisfiability problem. The weighted max-satisfiability problem is a NP-Complete problem [5]. Hence, the problem of finding the minimum and maximum currents drawn by a PLA is a hard problem. In [4], Devadas et al. use exact or approximate methods to solve the weighted max-satisfiability problem. The resulting algorithms are complex and the analysis is slow even for small circuits.

In our approach, we transform the problem of estimating maximum or minimum current in a PLA to a graph problem. We create a graph called the constraint graph and use graph-theoretic algorithms to find a solution to the graph problem. The description of the graph problems and the algorithms is presented in the subsequent sections. The number of vertices in the constraint graph is equal to the number of logic blocks in the circuit. There is a one-to-one correspondence between a vertex in the constraint graph and a logic block in the circuit. If two output nodes cannot simultaneously be set to LO for *any* input vector, then there exists an edge between the corresponding two vertices in the constraint graph. The weight associated with each vertex is related to the peak current drawn by the corresponding logic block. Although the weights associated with the vertices could be different, the constraint graphs for both the static and the dynamic pseudo-NMOS implementations of a circuit is the same. This is because the Boolean function of both the static and dynamic pseudo-NMOS implementations is the same. Hence, in this paper we present the analysis of dynamic pseudo-NMOS implementations. These methods are also valid for static pseudo-NMOS implementations. In a dynamic pseudo-NMOS implementation, the peak current is dependent on the conductance of the PMOS transistor and the total output node capacitance. Hence, if the peak current is not specified, then the weight of a vertex w_i corresponding to logic block i is computed using the following equation,

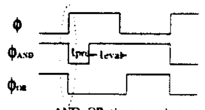

Figure 3: Clock signals for dynamic implementation of PLAs

$$w_i = k_i * \{C_1 * L_i + C_2 * N_i + C_3 * O_i\} \tag{1}$$

k_i is related to the conductance of the PMOS transistor of the logic block. C_1, C_2, C_3 are nominal capacitance values that are dependent on the technology parameters. C_1, C_2, C_3 are related to the interconnect capacitance, source/drain capacitance, and gate capacitance respectively. L_i denotes the interconnect length and hence $(C_1 * L_i)$ denotes the contribution of interconnect capacitance to the total capacitance of a logic block. N_i denotes the number of inputs to the logic block i and hence $(C_2 * N_i)$ denotes the contribution of drain/source capacitance to the total capacitance of a logic block. O_i corresponds to the fanout count of the output node of the logic block and hence $(C_3 * O_i)$ denotes the contribution of fanout gate capacitance to the total capacitance of a logic block.

3 Implementation Styles

Single level logic functions implemented using PMOS or NMOS networks are always inverting functions. Hence, there are only two possible implementations of the AND plane, using NAND or NOR functions. For the pseudo-NMOS implementation as shown in Fig. 2, the NMOS network consists of series connected NMOS transistors for NAND functions and parallel connected NMOS transistors for NOR functions. Each implementation has its advantages and disadvantages. The NAND implementation is slow, but it dissipates less power compared to a NOR implementation. The OR plane can also be implemented as a NAND or a NOR function. This results in four different implementations of an AND-OR PLA: NAND-NAND, NAND-NOR, NOR-NAND, NOR-NOR. In this paper, we analyze NAND-NAND and NOR-NOR implementations of AND-OR functions. The methodology we present is very general and it can be extended to the NAND-NOR or NOR-NAND implementations of an AND-OR PLA.

In the NAND-NAND or NOR-NOR dynamic pseudo-NMOS implementations the two planes cannot be directly cascaded. The AND plane and the OR plane must have different clock signals. We use the multiple clocking scheme presented in [6]. The clock signals for a dynamic pseudo-NMOS implementation are shown in Fig. 3. t_{pre} and t_{eval} represent the worst-case precharge and evaluation times of an AND row. In the precharge phase, ϕ_{AND} and ϕ_{OR} are LO and in the evaluate phase ϕ_{AND} and ϕ_{OR} are HI. In the precharge phase, all

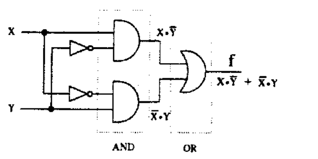

Figure 4: Two level AND-OR representation of XOR function

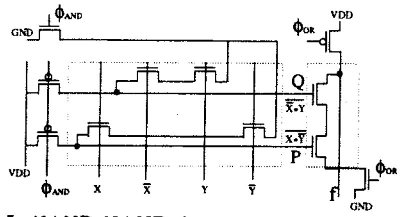

Figure 5: NAND-NAND dynamic impl. of XOR function

the logic blocks with the output node discharged due to the previous input vector draw simultaneous charging current. If two logic blocks cannot draw simultaneous charging current over all possible input vectors, then an edge is added between the corresponding vertices in the constraint graph. The construction of the constraint graph for a NAND-NAND implementation of a PLA is described in the next subsection.

3.1 NAND-NAND Implementation

In a NAND-NAND implementation, the product terms are generated by performing a NAND operation on a sub-set of PLA inputs and the complements of PLA inputs. The PLA outputs are generated by performing a NAND operation on a sub-set of the product terms. The product terms for the NAND-NAND implementation are complements of the corresponding product terms in the AND-OR sum of products representation. Consider the two level AND-OR representation of a two input XOR function shown in Fig. 4. The two level sum of products expression for the output $\mathbf{f}$ of the XOR function is given by $\mathbf{f} = \mathbf{X} \cdot \overline{\mathbf{Y}} + \overline{\mathbf{X}} \cdot \mathbf{Y}$. Using a few Boolean manipulations, $\mathbf{f}$ can be rewritten in the NAND-NAND format as $\mathbf{f} = \overline{\overline{\mathbf{X} \cdot \overline{\mathbf{Y}}} \cdot \overline{\overline{\mathbf{X}} \cdot \mathbf{Y}}}$. The NAND-NAND dynamic implementation of XOR function is shown in Fig. 5. Observe that the product terms are complements of the corresponding product terms in the AND-OR representation. When a NAND logic block corresponding to a product term evaluates to LO, it implies that all the input literals of the product term are HI. The complement input nodes of these input literals would be LO. Hence, all the product terms containing at least one of the complement input nodes cannot be LO. This is because the output of a NAND gate with at least one LO input node evaluates to a HI. Each product term and PLA output node corresponds to a vertex in the constraint graph. Since an edge is added between two vertices in the constraint graph if the corresponding logic block output nodes cannot be discharged (LO) simultaneously, the edges in the constraint graph are created using the following rules:

- If a product term P is LO, then a product term Q with at least one literal that is a complement of the literals present in P cannot be LO. Hence, there exists an edge between the vertices that correspond to (P, Q).
- If a product term P is LO, then a PLA output node O that contains the product term P as an input cannot be LO. Hence, there exists an edge between the vertices that correspond to (P, O).

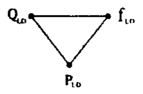

Figure 6: Constraint graph for NAND-NAND impl. of XOR

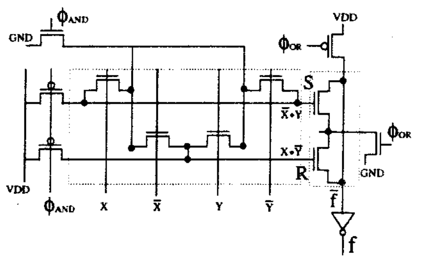

Figure 7: NOR-NOR dynamic implementation of XOR function

In Fig. 5 the nodes P and Q correspond to the product terms $\overline{\mathbf{X} \cdot \overline{\mathbf{Y}}}$ and $\overline{\overline{\mathbf{X}} \cdot \mathbf{Y}}$ respectively. The output node of the XOR function is denoted by $\mathbf{f}$ and it corresponds to $\overline{P \cdot Q}$. It can be seen that if any one of the terms $(P, Q, \mathbf{f})$ is LO, then the other two terms cannot be LO. Fig. 6 shows the constraint graph for the NAND-NAND implementation of XOR function. Using the methodology presented in this section, it is possible to construct the constraint graph for a dynamic NAND-NAND implementation of any AND-OR PLA. In the next subsection we formulate the problem for a dynamic NOR-NOR implementation of an AND-OR PLA.

3.2 NOR-NOR Implementation

In a NOR-NOR implementation, the product terms are generated by performing a NOR operation on a sub-set of PLA inputs and the complements of PLA inputs. The PLA outputs are generated by performing a NOR operation on a sub-set of the product terms. The product terms for the NOR-NOR implementation are the same as the corresponding product terms in the AND-OR sum of products representation. Consider the two level AND-OR representation of a two input XOR function shown in Fig. 4. Using a few Boolean manipulations, the output $\mathbf{f}$ of the XOR function can be rewritten in the NOR-NOR format as $\overline{\mathbf{f}} = \overline{\mathbf{X} \cdot \overline{\mathbf{Y}} + \overline{\mathbf{X}} \cdot \mathbf{Y}} = \overline{\overline{\overline{\mathbf{X}} + \mathbf{Y}} + \overline{\mathbf{X} + \overline{\mathbf{Y}}}}$. The NOR-NOR dynamic implementation of XOR function is shown in Fig. 7. Observe that the product terms are the same as the corresponding product terms in the AND-OR representation. The PLA outputs of the NOR-NOR implementation are complemented and hence, an additional inverter is required for each PLA output. If the output of a NOR logic block is LO, it does not imply any value at the inputs. If the input to a NOR logic block is LO, it does not imply any value at the output. Hence, it is not possible to construct a constraint graph by the method used for the NAND-NAND implementation. We now present the duality relationship between the NAND-NAND and NOR-NOR implementations of an AND-OR PLA. Observe that the product terms (and PLA outputs) in the NAND-NAND implementation and the NOR-NOR implementation are complements of each other. Hence, it can be seen that when an input vector sets the output of a logic block LO in the NAND-NAND implementation, the same input vector sets the output of the corresponding logic block HI in the NOR-NOR implementation. An input vector that *maximizes* the number of logic blocks set to LO in the NAND-NAND implementation also *maximizes* the number of logic blocks set to HI in the NOR-NOR implementation. The output node of a logic block can take only two logic values, either HI or LO. If an input vector sets the maximum number of logic blocks to HI in the NOR-NOR implementation then it implies that the same vector sets the minimum number of logic blocks to LO in the same circuit. Hence, the input vector that *maximizes* the number of logic blocks set to LO in the NAND-NAND implementation, *minimizes* the number of logic blocks set to LO in the NOR-NOR implementation. Similarly, the input vector that *minimizes* the number of logic blocks set to LO in the NAND-NAND implementation, *maximizes* the number of logic blocks set to LO in the NOR-NOR implementation. This duality relationship is used to find the maximum and minimum currents drawn by the NOR-NOR implementation of the PLA. The sequence of operations to compute the maximum and minimum currents for the NOR-NOR implementation are given below.

- Given a NOR-NOR PLA, construct the constraint graph for the corresponding NAND-NAND implementation and set the weights of the vertices as the peak current drawn by the logic blocks in the NOR-NOR implementation.
- Using the methodology presented in the next section find an estimate of the maximum and minimum current in the NAND-NAND implementation. Use the duality relationship to obtain the maximum and minimum currents for the NOR-NOR implementation.

In the next section, we present the algorithms that use the constraint graph to find a group of nodes that can be set to LO so that the current drawn is either the maximum or minimum.

4 Constraint Graph

In the constraint graph the weight associated with a vertex is related to the peak current drawn by the corresponding logic block in the circuit. In a pseudo-NMOS dynamic implementation, the node is pulled LO by discharging the output node capacitance. In the next cycle; current is drawn in the precharge phase to charge all the discharged output node capacitances. An edge between two vertices in the constraint graph implies that the two logic blocks corresponding to the vertices cannot draw simultaneous charging current over all possible input vectors. Finding a group of logic blocks that can draw simultaneous charging current is equivalent to finding a group of vertices in the constraint graph with no edges between any pair of vertices. A group of vertices for which there are no edges between any pair of vertices and the sum of the vertex weights is maximum gives the maximum current drawn by the PLA over all possible input vectors. This problem is exactly the problem of finding the maximum weight independent set in the constraint graph, where the weights correspond to current drawn by each logic block. An independent set in a graph is defined as a set of vertices with no edges between any pair of the vertices. A maximal independent set is an independent set for which none of the vertices in the remaining graph can be appended to increase the size of the independent set. A maximum weight independent set in a graph is a maximal independent set for which the sum of weights on the vertices is maximum over all maximal independent sets. The problem of determining the maximum weight independent set in an arbitrary graph is NP-Complete [5]. There exists some classes of graphs like perfect graphs, claw-free graphs for which the problem can be solved in polynomial time. Since the constraint graph does not belong to any of these special classes of graphs, it is not known if the maximum weight independent set problem can be solved in polynomial time for the constraint graph. We use two algorithms to estimate the maximum weight independent set in the graph. The first algorithm is a Greedy algorithm for **MAX**imum weight independent set (**GMAX**) in the constraint graph. It is an iterative algorithm that picks the vertex of maximum gain in the graph. The gain for each vertex is the value obtained by subtracting the sum of the weight of the vertices adjacent to the vertex from the weight of the vertex. The selected vertex and the vertices adjacent to it are then deleted to obtain the new subgraph. The iterative algorithm is applied on the new subgraph till all the vertices in the graph are deleted. The sum of the weights of the selected vertices gives an estimate of the maximum weight independent set in the constraint graph. The second algorithm is an Exact algorithm for **MAX**imum weight independent set (**EMAX**) in the constraint graph. We have implemented the recursive backtracking algorithm presented in [8] for finding the maximum weight independent set in a graph.

In pseudo-NMOS dynamic implementation of a PLA, logic blocks draw the charging current in the precharge phase only if the output node is LO (discharged). An input vector sets the output node of a group of logic blocks to LO and the other output nodes to HI. The minimum current drawn by a PLA corresponds to the minimum value of the sum of the peak current values of the logic blocks set to LO. This corresponds to a set of vertices in the constraint graph that form a maximal independent set such that the sum of the vertex weights is minimum. This is the problem of finding the minimum weight maximal independent set or the minimum weight dominating independent set in the constraint graph. In an unweighted graph, the problem of finding the minimum maximal independence number or the minimum dominating independence number for arbitrary graph is NP-Complete [7]. In this work, we use a Greedy algorithm for **MIN**imum weight maximal independent set (**GMIN**) in the constraint graph. It is an iterative algorithm that picks the vertex of maximum gain in the graph. The gain for each vertex is the value obtained by subtracting the weight of the vertex from the sum of the weight of the vertices adjacent to it in the graph . The selected vertex and the vertices adjacent to it are then deleted to obtain the new subgraph. The iterative algorithm is applied on the new subgraph till all the vertices in the graph are deleted. The sum of the weights of the selected vertices gives an estimate of the minimum weight maximal independent set in the constraint graph.

5 Experimental Results

The experimental results were obtained on the MCNC two-level benchmark circuits [9]. These two-level circuits were minimized using ESPRESSO [10]. The two-level benchmark circuits are in the AND-OR format. We create the constraint graph for the corresponding NAND-NAND implementation. The weight associated with each vertex in the constraint graph is related to the peak current drawn by the corresponding logic block and it is computed using Equation 1. The default values for C_1, C_2, C_3 were chosen as 0.1, 3, 10 units respectively. The default value of L_i which denotes the length of the interconnect is taken as the number of PLA inputs for the logic blocks in the AND plane and the number of product terms for the logic blocks in the OR plane of the AND-OR PLA. The values of N_i, O_i were taken as the number of inputs and the fanout count of the logic blocks. We use the same weights for both the NAND-NAND and NOR-NOR implementations of the PLA.

We use the following notation for presenting the results. W_{total} denotes the sum of the weights on all the vertices in the graph. W_{gmax} denotes the sum of the weights on the vertices obtained as the maximum weight independent set using the **GMAX** algorithm. W_{emax} denotes the sum of the weights on the vertices obtained as the maximum weight independent set using the **EMAX** algorithm. W_{gmin} denotes the sum of the weights on the vertices obtained as the minimum weight maximal independent set using the **GMIN** algorithm. I_{max} and I_{min} denote the maximum and minimum currents obtained using exhaustive simulation. In the exhaustive simulation, for each input vector we determine the sum of weights on vertices that were set LO. I_{max}, I_{min} are the maximum and minimum values of the sum over all input vectors. For circuits with large number of inputs, we report the I_{max} and I_{min} values obtained by simulation for 2^{16} random input vectors. For these circuits, the I_{max} and I_{min} values are a lower bound and upper bound estimates of the maximum and minimum currents respectively. The run-time is in CPU seconds on a Sun-Sparc20 workstation.

For a NAND-NAND implementation, W_{gmax} and W_{emax} are estimates of the maximum current. W_{gmin} is an estimate of minimum current. I_{max}, I_{min} are obtained either by exhaustive simulation or by simulation for 2^{16} random input vectors. Table 1 and Table 2 show the results for the maximum and minimum current drawn by the NAND-NAND implementation of a PLA. It can be seen that the algorithms are fast and generate accurate results. **GMAX** algorithm is fast and within a small amount of time it generates W_{gmax} which is a lower bound on the maximum current. **EMAX** algorithm generates W_{emax} which is equal to the maximum current value obtained by exhaustive simulation. For benchmark circuits with large number of primary inputs the **EMAX** algorithm generates W_{emax} value greater than the I_{max} value obtained by a simulation for 2^{16} random input vectors. This clearly shows that the graph based algorithm **EMAX** is superior to the simulation based methods. The CPU-time requirements of the **EMAX** algorithm are nominal. The **GMIN** algorithm generates W_{gmin} which is an upper bound on the minimum current. For some benchmark circuits this method generates tight upper bound results and for other benchmarks it results in a loose upper bound. The CPU-time requirements of the **GMIN** algorithm are extremely small.

We now present the results for the NOR-NOR implementation of the AND-OR PLA. We use the duality relationship between the NAND-NAND and the NOR-NOR implementation to obtain the estimates of the maximum and minimum current for a NOR-NOR implementation. If W_{gmin} denotes the estimate of minimum current obtained using the NAND-NAND implementation then, $(W_{total} - W_{gmin})$ is an estimate of the maximum current for the NOR-NOR implementation. Similarly, if W_{gmax}, W_{emax} are estimates of maximum current for a NAND-NAND implementation then $(W_{total} - W_{gmax})$ and $(W_{total} - W_{emax})$ are estimates of the minimum current for the NOR-NOR implementation. Table 3 and Table 4 show the results for the maximum and minimum current drawn by the NOR-NOR implementation of a PLA. The **GMIN** algorithm generates $(W_{total} - W_{gmin})$ which is a lower bound on the maximum current in a small amount of time. The results generated by this algorithm are close to the I_{max} values. **GMAX** algorithm is fast and within a small amount of time it generates $(W_{total} - W_{gmax})$ which is an upper bound on the minimum current. **EMAX** algorithm generates $(W_{total} - W_{emax})$ which is equal to the minimum current value obtained by exhaustive simulation. For benchmark circuits with large number of primary inputs the **EMAX** algorithm generates $(W_{total} - W_{emax})$ value lesser than the I_{min} value obtained by simulation for 2^{16} random input vectors. This clearly shows that the graph based algorithm **EMAX** is superior to the simulation based methods. The CPU-time requirements of the **EMAX** algorithm are nominal and the maximum run-time over all the benchmark circuits is less than 5 minutes. Hence, the methods presented in this paper generate good estimates of the maximum and minimum currents drawn by a PLA in a small amount of time.

6 Conclusion

In this paper, we presented efficient methods for finding the maximum and minimum current drawn by a PLA. We transform the problem of finding maximum and minimum current to a graph problem and use graph-theoretic algorithms to find the solution. We also presented the duality relationship between the NAND-NAND implementation and the NOR-NOR implementation of the PLA. We exploit this duality relationship to find the maximum and minimum currents drawn by a NOR-NOR PLA. Comparisons with exhaustive simulation show that the methods are accurate and fast.

References

[1] C. Small, "Shrinking devices put the squeeze on system packaging," *EDN*, vol. 39, no. 4, pp. 41–46, February 1994.

[2] J. R. Black, "Electromigration failure modes in aluminum metalization for semiconductor devices," in *Proceedings of the IEEE*, vol. 57, no. 9, pp. 1587–1594, September 1969.

[3] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI design*. 2nd ed. Reading, MA: Addison-Wesley, 1993.

[4] S. Devadas, K. Keutzer, and J. White, "Estimation of power dissipation in CMOS combinational circuits using Boolean function manipulation," *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 3, pp. 373–383, March 1992.

[5] M. R. Garey and D. S. Johnson, *Computers and Intractability*. New York, NY: W. H. Freeman and Company, 1979.

[6] J. M. Rabaey, *Digital integrated circuits*. Upper Saddle River, NJ: Prentice Hall, 1996.

[7] M. M. Halldorsson, "Approximating the minimum maximal independence number," *Information Processing Letters*, vol. 46, no. 4, pp. 169–172, June 1993.

[8] E. Loukakis and C. Tsouros, "An algorithm for the maximum internally stable set in a weighted graph," *International Journal of Computer Mathematics*, vol. 13, no. 2, pp. 117–129, 1983.

[9] S. Yang, *Logic synthesis and optimization benchmarks user guide, Version 3.0*. MCNC, Research Triangle Park, NC, 1991.

[10] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*. Boston, MA: Kluwer Acad. publishers, 1984.

Table 1: NAND-NAND Maximum Current Results

Circuit	Using GMAX		Using EMAX		I_{MAX}
	W_{gmax}	Time	W_{emax}	Time	
Z5xp1	407.0	0.09	498.4	0.82	498.4
Z9sym	266.6	0.09	266.6	0.82	266.6
apex1	4236.0	4.34	4354.4	26.23	4288.4*
apex2	6335.9	4.50	6338.9	91.57	3908.3*
apex3	4466.0	16.63	4480.4	68.89	4466.0*
apex4	5877.7	14.36	5877.7	23.32	5877.7
b12	232.8	0.07	260.0	0.09	260.0
bw	802.6	0.07	802.6	0.09	802.6
clip	589.3	0.16	709.6	5.29	709.6
con1	72.8	0.07	72.8	0.07	72.8
cps	4614.7	5.73	4752.6	23.59	4729.9*
duke2	985.0	0.42	1032.4	1.81	1026.2*
e64	829.5	0.28	829.5	0.82	682.5*
ex1010	2394.0	1.15	2394.0	33.02	2394.0
ex5	1648.4	0.73	1648.4	2.69	1648.4
inc	248.1	0.07	251.8	0.09	251.8
misex1	144.4	0.04	157.8	0.06	157.8
misex2	210.6	0.07	293.0	0.11	253.3*
misex3c	1023.1	1.52	1025.5	54.99	1025.5
pdc	2140.0	1.56	2234.9	16.06	2234.9
rd53	114.3	0.06	183.1	0.11	183.1
rd73	492.9	0.19	492.9	0.71	492.9
rd84	938.4	0.52	938.4	3.71	938.4
sao2	227.8	0.07	227.8	0.10	227.8
seq	6804.0	12.01	7222.2	110.54	6884.7*
spla	3443.0	8.09	3473.6	77.26	3473.6
squar5	157.0	0.06	164.0	0.08	164.0
t481	1491.1	2.52	2662.4	282.72	2662.4
table3	2174.0	0.85	2174.0	9.89	2174.0
table5	2055.0	0.79	2055.0	6.24	2055.0*
vg2	543.0	0.19	698.0	5.74	644.0*
xor5	49.6	0.05	49.6	0.07	49.6

* Not an exhaustive simulation (2^{16} random vector simulation)

Table 2: NAND-NAND Minimum Current Results

Circuit	Using GMIN		I_{MIN}
	W_{gmin}	Time	
Z5xp1	307.7	0.06	175.7
Z9sym	57.8	0.08	28.9
apex1	1959.3	0.28	1570.1*
apex2	673.9	0.45	336.9*
apex3	2121.8	0.64	2121.8*
apex4	1894.3	1.91	1787.2
b12	232.8	0.07	118.3
bw	380.7	0.05	380.7
clip	103.7	0.08	103.7
con1	66.0	0.06	28.8
cps	3180.4	0.19	3028.3*
duke2	739.2	0.07	684.2*
e64	630.5	0.07	617.5*
ex1010	1246.5	0.54	276.0
ex5	926.0	0.09	700.2
inc	182.7	0.06	154.7
misex1	83.2	0.06	83.2
misex2	157.3	0.05	150.1*
misex3c	827.3	0.20	437.8
pdc	1412.6	0.07	1151.6
rd53	53.6	0.06	53.6
rd73	51.7	0.09	51.7
rd84	76.4	0.23	76.4
sao2	71.0	0.06	71.0
seq	3461.1	1.96	3461.1*
spla	2682.6	0.49	2322.6
squar5	106.5	0.05	106.5
t481	214.4	1.36	23.0
table3	550.9	0.24	510.4
table5	501.5	0.17	407.8*
vg2	370.0	0.09	226.0*
xor5	25.5	0.07	25.5

* Not an exhaustive simulation (2^{16} random vector simulation)

Table 3: NOR-NOR Maximum Current Results

Circuit	Using GMIN		I_{MAX}
	(W_{total} - W_{gmin})	Time	
Z5xp1	2145.8	0.06	2277.8
Z9sym	2694.2	0.08	2723.1
apex1	19450.7	0.28	19839.9*
apex2	15448.5	0.45	16085.6*
apex3	20886.2	0.64	20886.2*
apex4	32893.6	1.91	33000.7
b12	705.2	0.07	727.3
bw	3208.9	0.05	3208.9
clip	4063.3	0.08	4063.3
con1	128.1	0.06	165.3
cps	16955.5	0.19	17105.8*
duke2	5137.4	0.07	5192.4*
e64	7494.5	0.07	7507.5*
ex1010	14245.5	0.54	15216.0
ex5	19898.4	0.09	20124.2
inc	1079.3	0.06	1107.3
misex1	672.8	0.06	672.8
misex2	1013.1	0.05	1020.3*
misex3c	6945.9	0.20	7335.4
pdc	8895.4	0.07	9156.4
rd53	846.2	0.06	846.2
rd73	3916.3	0.09	3916.3
rd84	8177.8	0.23	8177.8
sao2	1887.8	0.06	1887.8
seq	36587.5	1.96	36587.5*
spla	16610.8	0.49	16970.4
squar5	603.0	0.05	603.0
t481	21112.2	1.36	21302.9
table3	14301.1	0.24	14341.6
table5	13567.1	0.17	13660.8*
vg2	3835.0	0.09	3979.0*
xor5	432.1	0.07	432.1

* Not an exhaustive simulation (2^{16} random vector simulation)

Table 4: NOR-NOR Minimum Current Results

Circuit	Using GMAX		Using EMAX		I_{MIN}
	(W_{total} - W_{gmax})	Time	(W_{total} - W_{emax})	Time	
Z5xp1	2046.5	0.09	1955.1	0.82	1955.1
Z9sym	2485.4	0.09	2485.4	0.82	2485.4
apex1	17174.0	4.34	17055.6	26.23	17121.6*
apex2	9786.5	4.50	9783.6	91.57	13786.4*
apex3	18542.0	16.63	18527.6	68.89	18542.0*
apex4	28910.2	14.36	28910.2	23.32	28910.2
b12	612.8	0.07	585.6	0.09	585.6
bw	2787.0	0.07	2787.0	0.09	2787.0
clip	3577.7	0.16	3457.4	5.29	3457.4
con1	121.3	0.07	121.3	0.07	121.3
cps	15521.2	5.73	15383.3	23.59	15406.0*
duke2	4891.6	0.42	4848.2	1.81	4850.4*
e64	7295.5	0.28	7295.5	0.82	7436.5*
ex1010	13098.0	1.15	13098.0	33.02	13098.0
ex5	19176.0	0.73	19176.0	2.69	19176.0
inc	1013.9	0.07	1010.2	0.09	1010.2
misex1	611.6	0.04	598.2	0.06	598.2
misex2	959.8	0.07	877.4	0.11	917.1*
misex3c	6750.1	1.52	6747.7	54.99	6747.7
pdc	8168.0	1.56	8073.1	16.06	8073.1
rd53	785.5	0.06	716.7	0.11	716.7
rd73	3475.1	0.19	3475.1	0.71	3475.1
rd84	7315.8	0.52	7315.8	3.71	7315.8
sao2	1731.0	0.07	1731.0	0.10	1731.0
seq	33244.6	12.01	32826.4	110.54	33163.8*
spla	15850.0	8.09	15819.4	77.26	15819.4
squar5	552.5	0.06	545.5	0.08	545.5
t481	19835.5	2.52	18664.2	282.72	18664.2
table3	12698.0	0.85	12698.0	9.89	12698.0
table5	12013.6	0.79	12013.6	6.24	12013.6*
vg2	3507.0	0.19	3507.7	5.74	3558.5*
xor5	408.0	0.05	408.0	0.07	408.0

* Not an exhaustive simulation (2^{16} random vector simulation)