Automated Generation of Dual Rail Adiabatic Gates from Binary Decision Diagrams

Joseph Clark, Elijah Raffel, and Himanshu Thapliyal
Department of Electrical Engineering and Computer Science
University of Tennessee, Knoxville, TN, United States
jclar168@vols.utk.edu, eraffel@vols.utk.edu, hthapliyal@utk.edu

***Abstract*—Dual rail adiabatic circuit design offers hardware-level protection against side-channel power analysis attacks such as Differential Power Analysis (DPA) and Correlation Power Analysis (CPA) attacks. While considerable attention has been given to synthesizing logic tree-based adiabatic circuits, comparatively little attention has been given to generating truly secure circuit variants. This paper presents preliminary results for a secure dual rail adiabatic synthesis tool based on Binary Decision Diagrams (BDDs). The tool demonstrates encouraging performance in matching known optimal transistor counts for several basic logic gates, in addition to providing improvement over existing works on established benchmarks.**

***Index Terms*—Adiabatic logic, binary decision diagram, dual-rail pre-charge logic, logic gates.**

I. Introduction

Dual rail adiabatic circuitry is a circuit design paradigm which offers significant power savings and increased resilience against side-channel attacks compared with traditional CMOS circuitry. Many works have been published concerning the design automation of these circuits, generally focusing either on generation of gate or multiplexer based logic [1], [2], or on generation of logic trees which may be placed into the gate template for any of several adiabatic logic families [3]–[5].

We are interested in the synthesis of logic tree-based circuits due to their more compact nature, and we are specifically interested in using Binary Decision Diagrams (BDDs) due to the direct mapping between BDDs and dual-rail logic trees. Due to this direct mapping and the efficiency of the BDD representation, BDD-based synthesis algorithms are a very common method of synthesizing dual rail adiabatic logic. However, comparatively little work has been done to improve the security of the circuits produced by this approach. Indeed, although many works use BDD-based synthesis methods under the assumption that they are secure, producing a balanced, and therefore secure, circuit is deceptively difficult. Any asymmetry present in the circuit structure (such as varying circuit path lengths) can produce different power usage between input combinations, which renders the circuit vulnerable to side-channel attacks such as Correlation Power Analysis (CPA) attacks. One solution which addresses this shortcoming [6] modifies the BDD by inserting "dummy nodes" to equalize the path length over all branches. We have thus chosen this approach as a baseline, and have made some alterations to produce our preliminary results, namely adapting the method from producing Pass Transistor Logic (PTL) to static logic and improving the efficiency and transistor usage of the method.

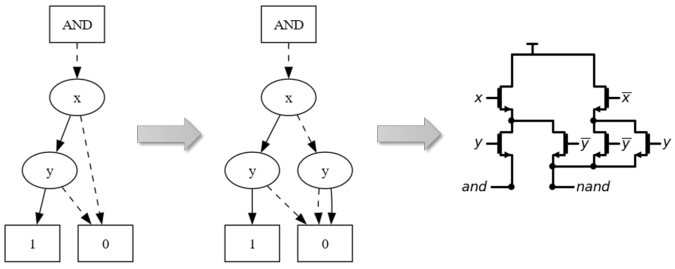


Fig. 1: BDD synthesis flow for an AND logic tree. The left hand side shows the initial BDD, to which dummy nodes are then added. The BDD with dummy nodes is then transformed into a logic tree by replacing each node with a differential pair of NMOS transistors (also understood as transforming each BDD graph edge into a transistor).

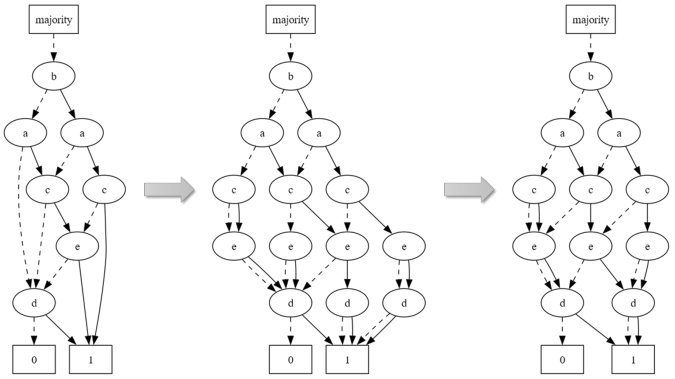
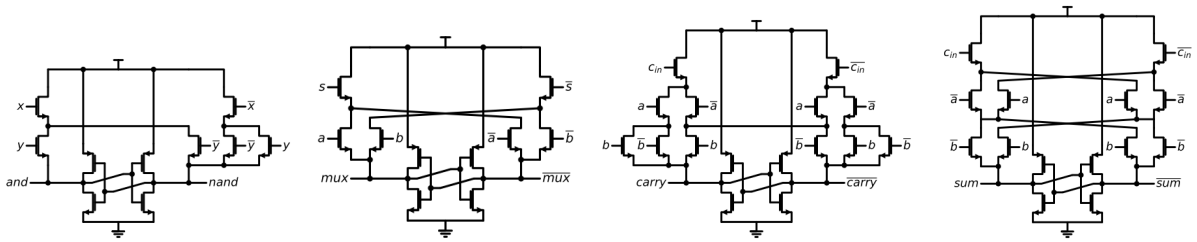


Fig. 2: Generation of dummy transistors for LGSynth91 majority circuit. The leftmost BDD is the initial ROBDD for the MAJ gate. The middle shows the suboptimal BDD structure which does not take advantage of dummy node reuse, while the BDD on the right shows the optimal structure which accounts for reuse.

II. Methods and Results

We implemented the BDD-based synthesis with path-balancing method in Python using the BDD package DD [7], which includes Cython bindings to well-known BDD packages such as CUDD [8]. Although our current implementation



(a) AND gate. (b) 2:1 multiplexer logic. (c) Full adder carry logic. (d) Full adder sum logic.

Fig. 3: Generated preliminary benchmark results as PFAL circuits.

requires input to be given in the form of a boolean logic expression in order to produce a BDD, we plan to integrate future versions directly with a Hardware Description Language such as Verilog or VHDL. The resulting BDD is then minimized into a Reduced Ordered Binary Decision Diagram (ROBDD) using Rudell's shifting algorithm [9]. After the ROBDD is created, is then transformed into a secure implementation of the original logic function by following the synthesis procedure outlined in [6] which balances the number of transistors along each circuit path. This process is shown in Figure 1.

To balance the BDD paths, we must ensure that each route from the top of the BDD to the bottom passes through the same number of nodes. This is accomplished by inserting "dummy" nodes, which have both of their branches connected to the same node along the original path. This has the additional effect of ensuring the number of transistors connected to the high and low rails of each variable is balanced. The BDD is then converted into an NMOS logic tree by converting each BDD variable node into a differential pair of NMOS transistors, the starting node into the source voltage rail, and the true and false outputs into the high and low rails of the output, respectively. This logic tree can then be inserted into the circuit template for most adiabatic logic families to produce an adiabatic logic circuit. It should be noted that we are currently focusing on synthesis of static logic rather than Pass Transistor Logic (PTL) due to the reduction in transistor counts derived from combining the high and low logic trees; as a result, we connect our circuit in the opposite direction of [6], with the source at the top rather than the output.

To improve the efficiency of the path balancing algorithm, we have made two major changes. First, we have restructured the algorithm into two major steps: finding the path length for each node (longest path) and then performing a graph traversal to insert the required dummy nodes. The longest path from a set of nodes to all other nodes in a Directed Acyclic Graph (DAG) can be calculated in linear time by performing a post-order traversal of the BDD and keeping the maximum of the child node distances as the node distance when visiting a node. Next, we traverse the graph and insert dummy nodes to balance each pair of children (for example, if one child has distance 5, and the other has distance 3, 2 dummy nodes are inserted in the second path). Second, we have optimized the count of dummy nodes by keeping track of inserted dummy nodes and reusing them when possible, an example of which is shown in Figure 2. This is accomplished using a hash table which is keyed on the desired endpoint node and the number of remaining dummy transistors to be inserted along the path, and which stores a reference to the dummy node inserted at that location.

TABLE I: Comparison of transistor counts between the synthesis tool and the best known dual rail implementation of various logic function trees.

Function	# in	Number of Transistors		
		Existing	This Work	w/Balance
AND	2	4	4	6
OR	2	4	4	6
XOR	2	6	6	6
MUX	2	6	6	6
FA Carry	3	8	8	12
FA Sum	3	10	10	10

TABLE II: Performance comparison of balanced dual rail adiabiatic logic trees for the PRESENT substitution box. Existing transistor counts are referenced from [6].

Function	Number of Transistors	
	Existing	This Work
S_0	18	18
S_1	24	22
S_2	22	22
S_3	22	22

Transistor counts of the logic trees produced for several basic logic functions by this synthesis method are shown in Table I. As expected, if the dummy connections are ignored, the tool matches the optimal number of transistors for the static dual rail implementation of each logic function: 4 for AND/OR, 6 for XOR, 6 for 2:1 multiplexer, 8 for Full Adder

TABLE III: Performance of synthesis tool on various small benchmark circuits. Benchmark name, number of inputs and outputs, and logic tree transistor count with and without balancing and with and without dummy node reuse are shown.

Function	# in	# out	Number of Transistors		
			Base	Balanced	w/Reuse
b1	3	4	20	24	22
CM82	5	3	40	52	52
CM85	11	3	78	298	180
CM151	12	2	64	80	80
CM152	11	1	30	30	30
majority	5	1	14	26	22
z4ml	7	4	86	162	122

(FA) Carry, and 10 for FA Sum (3-input XOR). Additionally, as expected, the tool does not insert dummy transistors for the 2 and 3-input XOR circuits or the 2:1 multiplexer circuit, as these circuits are already balanced. The AND/OR circuits have 2 transistors added, which is the minimum that can be added to balance the circuit, as transistors must be added in pairs and only one path in each circuit is too short. However, the FA carry circuit has four transistors added, which could be improved.

Circuits for three of the benchmark functions are shown in Figure 3, implemented as PFAL [10] logic. Although the AND gate does not use the standard PFAL structure, it is more easily balanced than the standard version, as only one path is out of balance (compared with two in the original). The carry circuit has 12 transistors, which does produce a balanced circuit, but could have been reduced to only 10 while still balancing the transistor counts, as the a transistors along of the each outside paths could simply have been duplicated to balance the paths. The multiplexer and sum circuits are identical to the known optimal 2:1 multiplexer and 3-input XOR circuits, respectively. We have also compared the quality of the balanced PRESENT substitution box circuits generated by our tool and those presented in [6]. The results are shown in Table II. As expected, our results match the existing ones, with our tool even achieving a slight savings on bit 1 of the output.

In order to evaluate the impact of dummy node reuse on generated circuits, we have measured the performance of the tool on several of the smaller combinational LGSynth91 benchmark circuits. The results are shown in Table III. From the table, it is clear that while the dummy node reuse has little or no effect in some cases, it makes a substantial difference in others. For example, $cm82a$, $cm151a$, and $cm152a$ show no change with resuse, while $b1$ and $majority$ see a small improvement of one or two nodes saved. The two largest circuits, $cm85a$ and $z4ml$, on the other hand, see an improvement of 118 and 40 transistors, respectively, constituting a 40% and 25% reduction in the total number of transistors.

III. Discussion and Conclusion

BDD-based synthesis provides an efficient baseline synthesis method for dual rail adiabatic static logic. Our preliminary work is shown to match optimal solutions for important fundamental circuits such as XOR gates and adder logic, while matching the transistor count for the usual implementation of other circuits, such as AND/OR gates. Additionally, by adding a few redundant nodes to the BDD, the number of transistors along each circuit path can be reliably balanced to provide secure computation. Our tool is also shown to match the performance of existing work in this area on the PRESENT substitution box, and outperform existing methods for several benchmark circuits due to our improved reuse of dummy transistors during balancing.

Our future work on this tool will consist of streamlining and optimizing the existing approach. This includes tight integration with Verilog, which existing tools generally disregard in favor of modifying already-synthesized circuits to produce dual rail variants. We plan to optimize the tool by improving the existing synthesis and path balancing approaches, in addition to exploring other synthesis methods and new path balancing algorithms. We will also implement several security features, such as the ability to analyze the security of circuits produced by the tool.

Acknowledgments

This work was supported in part by the National Science Foundation CAREER Award under Grant 2232235.

References

[1] G. Paul, S. N. Pradhan, A. Pal, and B. B. Bhattacharya, "Low power bdd-based synthesis using dual rail static dcvspg logic," in *APCCAS 2006-2006 IEEE Asia Pacific Conference on Circuits and Systems*, IEEE, 2006, pp. 1504–1507.

[2] S. N. Pradhan, G. Paul, A. Pal, and B. B. Bhattacharya, "Power aware bdd-based logic synthesis using adiabatic multiplexers," in *2006 International Conference on Electrical and Computer Engineering*, IEEE, 2006, pp. 149–152.

[3] P. De, U. Parampalli, and C. Mandal, "Secure path balanced bdd-based pre-charge logic for masking," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4747–4760, 2020.

[4] R. Chaudhry, T.-H. Liu, A. Aziz, and J. L. Burns, "Area-oriented synthesis for pass-transistor logic," in *Proceedings International Conference on Computer Design. VLSI in Computers and Processors (Cat. No. 98CB36273)*, IEEE, 1998, pp. 160–167.

[5] T. Karoubalis, G. P. Alexiou, and N. Kanopoulos, "Optimal synthesis of differential cascode voltage switch (dcvs) logic circuits using ordered binary decision diagrams (obdds)," in *Proceedings of EURO-DAC. European Design Automation Conference*, IEEE, 1995, pp. 282–287.

[6] P. De, C. Mandal, and U. Prampalli, "Path-balanced logic design to realize block ciphers resistant to power and timing attacks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 5, pp. 1080–1092, 2019.

[7] I. Filippidis, S. Haeseart, S. C. Livingston, and M. Wenzel, *Dd*, https://github.com/tulip-control/dd, 2024.

[8] F. Somenzi, "Cudd: Cu decision diagram package release 2.3. 0," *University of Colorado at Boulder*, vol. 621, 1998.

[9] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, IEEE, 1993, pp. 42–47.

[10] A Vetuli, S. Di Pascoli, L. Reyneri, *et al.*, "Positive feedback in adiabatic logic," *Electronics Letters*, vol. 32, no. 20, pp. 1867–1868, 1996.