The Analysis of Cyclic Circuits with Boolean Satisfiability

John Backes, Brian Fett, and Marc D. Riedel

Department of Electrical and Computer Engineering
University of Minnesota
200 Union St. S.E., Minneapolis, MN 55455
{back0145, fett, mriedel}@umn.edu

***Abstract*—The accepted wisdom is that combinational circuits must have *acyclic* (i.e., loop-free or feed-forward) topologies. And yet simple examples suggest that this need not be so. In previous work, we advocated the design of *cyclic* combinational circuits (i.e., circuits with loops or feedback paths). We proposed a methodology for synthesizing such circuits and demonstrated that it produces significant improvements in area and in delay. The analysis method that we used to validate cyclic circuits was based on binary decision diagrams. In this paper, we propose a much more efficient technique for analysis based on Boolean satisfiability (SAT).**

I. Introduction

A. Cyclic Combinational Circuits

A collection of logic gates forms a *combinational* circuit if the outputs can be described as Boolean functions of the current input values only. A common misconception is that combinational circuits must have acyclic topologies; that is to say, they must be designed without any loops or feedback paths. In fact, the idea that "combinational" and "acyclic" are synonymous terms is so thoroughly ingrained that many textbooks provide the latter as a definition of the former (e.g., (1), p. 14; (2), p. 193)

Indeed, any acyclic circuit is clearly combinational. Regardless of the initial values on the wires, once the values of the inputs are fixed, the signals propagate to the outputs. The behavior of a circuit with feedback is generally more complicated. Such a circuit may exhibit sequential behavior, as in the case of an S−R latch, or it may be unstable, as in the case of an oscillator.

And yet, circuits with cyclic topologies can be combinational. Consider the example in Figure 1. It is combinational in the strictest sense: it produces the required output values *regardless* of the prior values on the wires and for *any* choice of delay parameters. If $x = 0$ then g_1 produces an output of 0, because 0 is a controlling value for an AND gate. If $x = 1$ then g_4 produces a value of 1, because 1 is a controlling value for an OR gate. In both cases, the cycle is broken and the circuit produces definite outputs. Since x must assume one of these two values, we conclude that the circuit *always* produces definite outputs. In fact, it implements two functions that both depend on all five variables:

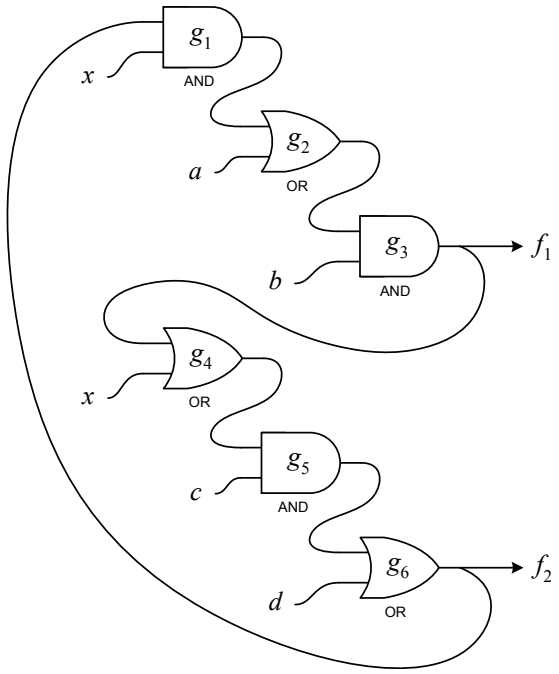

Fig. 1. A cyclic combinational circuit.

$$f_1 = b(a + x(d + c)),$$
$$f_2 = d + c(x + b\,a)$$

(+) denotes OR, (·) denotes AND

Note that the computation of the two functions overlaps. If we were to implement these functions with an acyclic circuit, we would need eight two-input gates.

B. Analyzing Cyclic Circuits

In previous work, we showed that combinational circuits can be optimized significantly if cycles are introduced (3), (4). A pivotal step in the synthesis methodology is determining whether cyclic circuits that are found behave combinationally.

This research has been funded in part by a grant from the SRC Focus Center Research Program on Functional Engineered Nano-Architectonics (FENA), contract No. 2003-NT-1107.

This analysis problem is conceptually straight-forward: correctness is ascertained by following all controlling values as they propagate through the circuit from the primary inputs – zeros controlling the outputs of AND gates, ones controlling the outputs of OR gates, these values controlling other gates, and so on. Of course, stepping through all possible input assignments is not a tractable proposition for real circuits: given n primary inputs, there would be 2^n input assignments to consider.

This is a specific problem but one that shares many properties with a broad class of problems in logic verification: it has an affirmative answer if a property holds for *all* possible input assignments; it has a negative answer if the property does not hold for *any* input assignment. The property – in this case, whether the circuit produces combinational behavior or not – is one directly ascribed to logical operations on the circuit – in this case, how controlling values propagate.

So-called SAT-based techniques, based on heuristic solutions to the Boolean satisfiability problem, have been deployed very successfully for problems in this vein (5), (6). Consider the classic problem in circuit verification: determining whether two circuits A and B are equivalent in the sense that they implement the same Boolean function. To solve this problem, one creates a new circuit C with the outputs of A and B tied together by an exclusive-OR gate. Then one asks the SAT question: is there some assignment of input values that *satisfies* the Boolean function implemented by C (i.e., for which the output of C evaluates to one)? If not, then the two circuits are equivalent. The starting point for SAT-based verification, then, is a circuit that returns identically zero (*UNSAT*) for an affirmative answer to the problem; and not identically zero (*SAT*) for a negative answer. The analysis proceeds by packaging the Boolean function implemented by the circuit as a formula in Conjunctive Normal Form (CNF). This is passed to heuristic algorithms known as SAT-Solvers (7), (8). In theory, such algorithms can take time that is exponential in the number of variables to complete. In practice, they have shown themselves to be remarkably efficient for problems in circuit verification, often handling large problem instances with thousands of variables with ease.

Of course, in order to package an analysis problem as a CNF formula for SAT, the starting point must be an *acyclic* circuit. Given a *cyclic* circuit, how can the analysis for combinationality proceed? We adopt a straight-forward yet efficient strategy.

- We find a *feedback arc set*, that is to say, wires that we can cut to make the circuit acyclic.
- We introduce new *dummy variables* at these cut locations.
- We encode the entire computation of the circuit in terms of ternary-valued logic: zeros, ones and "undefined" values. These ternary values are encoded with "dual-rail" binary values: zero is encoded as $[0, 0]$, one as $[1, 1]$, and "undefined" as either $[1, 0]$ or $[0, 1]$.
- We set up an acyclic circuit that answers the question: given undefined values for the dummy variables (in the ternary encoding) is there any input assignment that produces undefined values (again in the ternary encoding) at the output? This circuit forms the SAT question.

The algorithm is described in detail in Section II. The complexity is entirely dependent on the runtime of the SAT solver. Setting up the circuit for the SAT instance is comparatively trivial: it entails but a single pass through the circuit to compute a feedback arc set. The circuit for the SAT question is larger than the original circuit: for every gate in the original circuit, approximately six gates are needed to formulate the ternary-valued encoding; in addition to the primary inputs, the dummy variables at the cut locations are included. Given the efficiency of SAT solvers, this is a winning strategy in spite of the increase in the number of variables. In Section IV, we compare runtimes on benchmark circuits for this method compared to BDD-based methods.

C. Prior and Related Work

In an earlier era, theoreticians commented on the possibility of having cycles in combinational logic and conjectured that this might be a useful property (9), (10), (11). Both McCaw and Rivest presented examples of cyclic circuits with provably fewer gates than is possible with equivalent acyclic circuits (12), (13). (We have extended and generalized these theoretical results. Most notably, we have constructed a family of circuits with cyclic topologies having half as many gates as is possible with acyclic topologies (4)).

In a later era, practitioners observed that cycles sometimes appear in combinational circuits synthesized from high-level descriptions. Stok noted that cycles can be introduced during resource-sharing optimizations at the level of functional units (14). However, since synthesis and verification tools balk when given combinational logic with cycles, he concluded that those optimizations have to be rejected at the high-level phase.

Motivated by Stok's observation, Malik discussed analysis techniques for cyclic circuits (15). He formulated a symbolic analysis algorithm based on ternary-valued simulation. He proposed a topological approach, beginning with a transformation from a cyclic specification to an equivalent acyclic one. Edwards followed a similar strategy, discussing techniques specifically targeted at cyclic circuits that are produced inadvertently during high-level design (16). Shiple refined and formalized Malik's results and extended the concepts to combinational logic embedded in sequential circuits (17).

In previous work, we described a methodology for synthesizing cyclic circuits (3). Our approach for synthesis is conceptually general. Cycles are introduced through the incremental application of restructuring and minimization operations, optimizing a design for area and delay. These optimizations are carried through to the decomposition and technology mapping phases. The methodology is implemented as a package called CYCLIFY, built within the Berkeley SIS environment (18). Trials on benchmark circuits as well as examples from industry demonstrated that cyclic solutions are not a rarity; they can readily be found for most circuits of practical interest. CYCLIFY reduced the area of standard benchmark circuits by as much as 30% and the delay by as much as 25%. For analysis, we discussed techniques for validating cyclic circuits based on symbolic event propagation with binary decision diagrams (BDDs) (19). We also discussed techniques performing timing analysis of cyclic circuits (20).

D. Circuit Model

The concepts discussed in this paper are not tied to any particular physical model or computing substrate. Generally the exposition is at a *symbolic* level, that is to say, in terms of Boolean expressions. However, we first discuss the circuit model in an explicit sense – in terms of signal values.

We work with the digital abstraction of zeros and ones. Nevertheless, our model recognizes that the underlying signals are, in fact, analog: each signal is a continuous real-valued function of time, corresponding to a voltage level. For analysis, we adopt a *ternary* framework, extending the set of *Boolean* values $\mathbb{B} = \{0, 1\}$ to the set of ternary values $\mathbb{T} = \{0, 1, \perp\}$. Here $\perp$ represents either an *ambiguous* value, e.g., a voltage value between logical 0 and logical 1, or else an *uncertain* value, i.e., a signal that might be 0 or 1 – but we do not know which.

The idea of three-valued logic for circuit analysis is well established. It was originally proposed for the analysis of *hazards* in combinational logic (21). Bryant popularized its use for verification (22), and it has been widely adopted for the analysis of asynchronous circuits (23). For a theoretical treatment, see (24). Malik and Shiple discuss the analysis of cyclic circuits in this framework (15), (17).

Central to the analysis is the concept of *controlling* values. In (4), a formalism is presented for computing the controlling values of arbitrary logic functions, in a symbolic context. For simplicity, in this paper we assume that the network has been decomposed into primitive gates, namely AND/OR/NAND/NOR gates and inverters. Recall that 0 is the controlling value for an AND gate, as shown in Figure 2. Similarly, 1 is the controlling value for an OR gate.

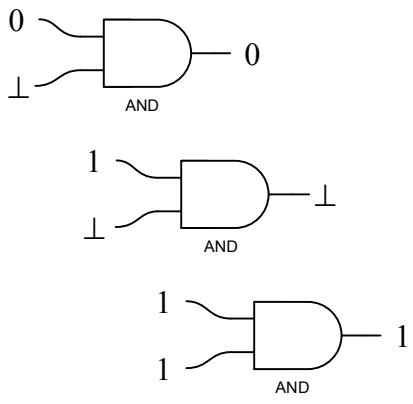

Fig. 2. An AND gate with 0, 1, and $\perp$ inputs.

Our analysis characterizes the functional and temporal behavior of circuits according to the so-called "floating-mode" assumption (23), (25): at the outset, all wires in a circuit are assumed to have unknown or possibly undefined values, and so are assigned the value $\perp$. This assumption ensures that the analysis does not infer stability in cases where ambiguous or unstable signals might persist.

Consider the circuit fragment in Figure 3. One might be tempted to reason as follows: the output of the AND gate g_1 is fed in complemented and uncomplemented form into the OR gate g_2. Thus, one of the inputs to the OR gate must be 1, and so its output must be 1.

And yet, by definition, $\perp$ designates an *undefined* value. For instance, it could indicate a voltage value exactly half way between logical 0 and logical 1. Within the floating-mode framework, we remain agnostic: the output of the OR gate is $\perp$.

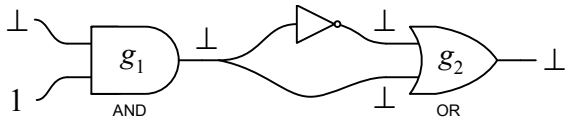

Fig. 3. An illustration of the floating mode.

Conceptually, the analysis that we perform for cyclic circuits is just an algorithmic implementation of the idea illustrated in Example 1. All the wires initially have value $\perp$. We apply definite values to the inputs, and track the propagation of well-defined signal values. Once a definite value is assigned to an internal wire, this value persists for the duration (so long as the input values are held constant). For any input assignment, a circuit reaches a so-called *fixed point* in the ternary framework: a state where no further updates of controlling values are possible. This fixed point is unique (23).

We define the *validity* of a cyclic circuit as follows:

- If, for some assignment to the primary inputs, there are $\perp$ values in the fixed point that the circuit settles at, then the circuit is "**Invalid**."
- Conversely, if for every assignment to the primary inputs there are no $\perp$ values in the fixed point that it settles at, then the circuit is "**Valid**."

Of course, if there are "don't-care" conditions, then validity only applies to assignments in the "care" set. We could adopt a less stringent definition, only insisting that no $\perp$ values persist at the primary outputs; this would not alter our algorithm materially, so here we use the more stringent definition that no $\perp$ values can persist on *any* of wires in the circuit, whether these be internal or at the primary outputs.

II. Algorithm

Given a cyclic circuit, the objective of the analysis is to produce an acyclic circuit that computes an output value that is identically zero if and only if the cyclic circuit is valid. This acyclic circuit will then be fed into a SAT solver; we will refer to it as the "SAT circuit".

1) The first step is to find wires that, if cut from the circuit, would **break all the cycles**. Such a set can be found through a simple depth-first search (26).

Bit 0	Bit 1	Value
0	0	0
0	1	$\perp$
1	0	$\perp$
1	1	1

Fig. 4. Dual-rail encoding scheme for ternary values.

2) The next step is to convert every gate in the circuit into a corresponding module that operates on the **dual-rail**

encoded ternary logic. Using the encoding scheme given in Figure 4, this step is straight-forward. Consider the encoding for an AND operation on ternary-valued inputs a and b. We use pairs of inputs for each value: a_0 and a_1 corresponding to a, and b_0 and b_1 corresponding to b. The outputs are encoded by the functions:

$$\begin{aligned} f_0 &= a_0 b_0 + a_1 b_0 \bar{b}_1 \\ f_1 &= a_1 b_1 + a_0 \bar{b}_1 \bar{b}_0 \end{aligned}$$

Other gates, such as OR, NAND, NOR, etc., can be implemented similarly. The NOT operation is particularly easy – we simply complement the bit on each rail.

3) Each **primary input** is simply considered twice to obtain its dual-rail encoding. This way, if the primary input is assigned logic 1, the value (11) is fed; if it is assigned logic 0 the value (00) is fed.
4) At every cut location, we introduce a pair of **dummy variables** feeding into the corresponding dual-rail module. This allows for the possibility that the value in the circuit is $\perp$, encoded as different values assigned to each of the dummies, (01) or (10).
5) For every pair of dummy variables, we set up an **equivalence checker**: this is a module that evaluates to 1 if and only if the value assigned to dummies agrees with the value computed by the circuit at the cut location. The circuit may be computing $\perp$, encoded as (01) or (10); in this case, the equivalence checker evaluates to 1 if the dummies have *different* values. Call the output of the equivalence checker x_i for each cut location i. For dummy variables d_1 and d_2 and gate outputs f_1 and f_2, the logic for the equivalence checker is

$$\begin{aligned} x_i = {} & \bar{d}_1 \bar{d}_2 \bar{f}_1 \bar{f}_2 + d_1 d_2 f_1 f_2 + \\ & \bar{d}_1 d_2 \bar{f}_1 f_2 + \bar{d}_1 d_2 f_1 \bar{f}_2 + \\ & d_1 \bar{d}_2 \bar{f}_1 f_2 + d_1 \bar{d}_2 f_1 \bar{f}_2. \end{aligned}$$

6) For every pair of dummy variables, we set up a **$\perp$-checker**: this is simply an exclusive-OR gate on the two dummies; it evaluates to 1 if and only if the dummies are assigned different values (and so the are encoding $\perp$). Call the output of the $\perp$-checker y_i for each cut location i.

   Note that rather than introducing dummy variables, equivalence checkers, and $\perp$-checkers into the SAT circuit, we could instead append the logically equivalent clauses to the circuit's CNF formula representation to produce the same results. By introducing dummy variables and equavalence gates into the SAT circuit, we are implicitly adding these clauses to the CNF formula. Many modern SAT techniques take advantage of circuit structure alongside the circuit's CNF representation in order to find a result faster (27). Using the later method would not make use of the structurial information that dummy variables, equivalence checkers and $\perp$-checker add to the circuit.
7) Finally, as illustrated in Figure 5, the output of the circuit is the AND of the AND of the x_i's and the OR of the y_i's.

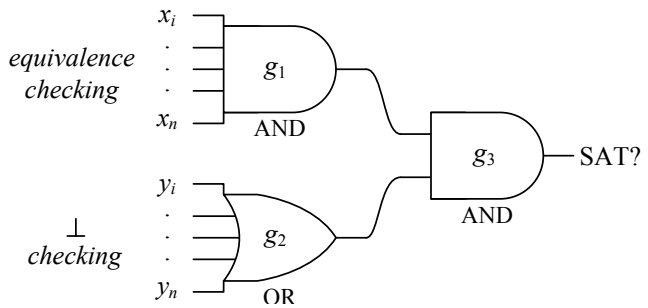

Fig. 5. Constructing the SAT instance.

Example 1

Consider the circuit in Figure 6, consisting of four NAND gates. Note that there are two cycles. Cutting these and inserting dummy variables d and e, we obtain the circuit in Figure 7. Next, we replace each gate with a dual-rail version; we feed in pairs of dummy variables, d_0, d_1, and e_0, e_1, corresponding to each of the previous dummy variables; we double up the primary inputs a and b; we add two equivalence-checkers, producing x_0 and x_1; we add two $\perp$-checkers (i.e., exclusive-OR gates) producing y_0 and y_1; and we add three logic gates g_1, g_2, and g_3 to form the final output.

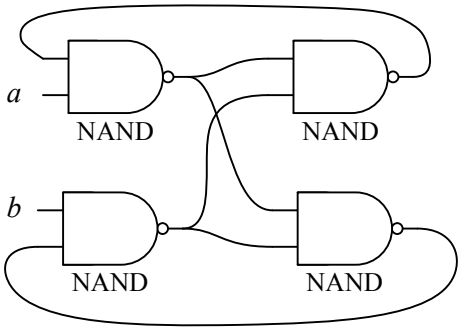

Fig. 6. A cyclic circuit

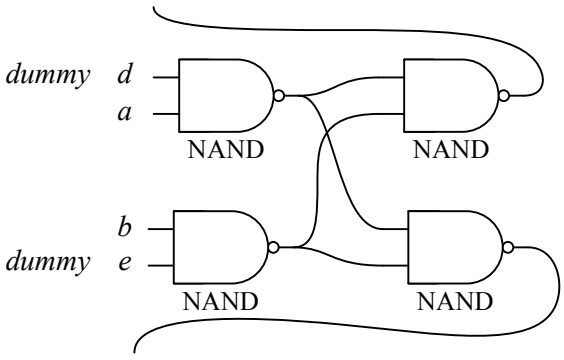

Fig. 7. The circuit in Figure 6 with cycles broken.

This circuit, shown in Figure 8, forms the SAT instance with six variables: a, b, d_0, d_1, e_0, and e_1. We see that for $a = b = 1$, $d_0 = \bar{d}_1$, and $e_0 = \bar{e}_1$, we get $\perp$ values on each pair of rails into the equivalence checkers, indicating that the inputs

to each are equivalent; so x_0 and x_1 produce outputs of 1; y_0 and y_1 produce outputs of 1 as well; so the final output is 1. Therefore, the SAT instance is *satisfiable* and so the circuit is *invalid*. Indeed, $a = b = 1$ are non-controlling values for the NAND gates, so this is the outcome that we expect. □

III. Proof of Correctness

First, we argue that a SAT circuit that evaluates to 1 never corresponds to a *valid* cyclic circuit. Indeed, if a SAT circuit evaluates to 1, then both the gates g_1 and g_2 are at 1. If g_1 is at 1, then the corresponding values in the cyclic circuit are at a *fixed point*; however, if g_2 is at 1, then some of the values in the fixed point are $\perp$. By definition, the cyclic circuit is invalid.

Next we argue that every invalid cyclic circuit translates into a SAT circuit that evaluates to 1 for a specific input assignment. Indeed, if the circuit is invalid then it has a fixed point with $\perp$ values on some of the wires of the cut set. (A fixed point that contains $\perp$ values *somewhere* must also have these on the cut set.) In the SAT circuit, consider such an input assignment: assign the dummy values that correspond to the values from the fixed point; this ensures that g_1 is at 1. Because some of these values are $\perp$, g_2 is also at 1 and so the SAT circuit evaluates to 1.

IV. Implementation and Results

We have implemented the algorithm described in Section Section II in the Berkeley ABC environment (28). ABC invokes the "MiniSat" SAT Solver (29). We performed trials on cyclic circuits produced by our tool, CYCLIFY, from benchmark circuits in the IWLS collection. (For circuits with latches, we extracted the combinational part.) All circuits had been mapped to 2-input NAND and NOR gates and inverters. We count the area of the NAND/NOR gates as 2, and that of inverters as 1. We compare the runtimes for the new SAT-based method to those using our previous BDD-based approach (20). Trials were performed on a 2.93 GHz Intel Core 2 Duo Processor running Linux.

V. Discussion

Early work in the 1960's and 70's established the premise of combinational circuits with cycles, and suggested the possible benefits. Still, combinational circuits are not designed with cycles in practice. Perhaps designers have eschewed feedback due to the apparent complexity of reasoning about cyclic structures. And yet, feedback provides significant opportunities for optimization, both for area and for delay. Indeed, contrary to the conventional wisdom, cyclic solutions are not a rarity; they can readily be found for most circuits that are not trivially simple or sparse. We have run trials with our program, called CYCLIFY, on a range of randomly generated examples and benchmark circuits. We note that solutions for most of the examples have deeply nested loops, with dozens or even hundreds of cycles.

Our synthesis strategy is to introduce feedback in the restructuring and minimization phases. A branch-and-bound search is performed, with analysis used to validate and rank potential solutions. Using BDDs, the analysis portion completely dominated the running time of the program CYCLIFY. The SAT-based methodology proposed here can tackle much larger benchmark circuits and it runs orders of magnitudes faster (as anyone familiar with SAT-based methods might have expected). For all of the circuits that we have tried, the time that it took to convert the original circuit description into the SAT instance took less than .01 seconds. Future work will include developing specific heuristics for finding smaller feedback arc sets; improvements here might positively impact the runtime of the algorithm. We are working on integrating the SAT-based method with synthesis; we will report the results in the near future.

		Runtimes (seconds)		
Circuit	Area	BDD Based	SAT Based	Ratio
5xp1	218	0.10	0.01	10.00
bbara	135	0.01	<0.01	1.00
clip	292	0.09	0.01	9.00
cse	346	0.13	0.03	4.33
dk16	426	0.09	0.03	3.00
duke2	664	2.35	0.07	33.57
ex1	514	0.36	0.07	5.14
keyb	401	0.24	0.03	8.00
misex3	1065	19.05	0.16	119.00
planet	890	1.03	0.08	12.88
planet1	882	1.40	0.11	12.73
pma	388	0.13	0.02	6.50
s1	555	0.56	0.06	9.33
s1488	1036	1.43	0.13	11.00
s386	224	0.02	0.02	1.00
sand	807	3.15	0.07	45.00
average	552	1.88	0.06	18.22

Fig. 9. Comparison of the runtime for the new SAT-based method with the older BDD-based method on cyclic circuits from the IWLS Benchmarks.

References

[1] R. Katz, *Contemporary Logic Design*. Benjamin/Cummings, 1992.

[2] J. F. Wakerly, *Digital Design: Principles and Practices*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2000.

[3] M. Riedel and J. Bruck, "The synthesis of cyclic combinational circuits," 2003. [Online]. Available: citeseer.ist.psu.edu/riedel03synthesis.html

[4] M. Riedel, "Cyclic combinational circuits," Ph.D. dissertation, 2004.

[5] H.-M. Koo and P. Mishra, "Test generation using sat-based bounded model checking for validation of pipelined processors," in *GLSVLSI '06: Proceedings of the 16th ACM Great Lakes symposium on VLSI*. New York, NY, USA: ACM, 2006, pp. 362–365.

[6] T. Larrabee, "Test pattern generation using boolean satisfiability," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 11, no. 1, pp. 4–15, Jan 1992.

[7] E. Goldberg and Y. Novikov, "Berkmin: A fast and robust sat solver," 2002. [Online]. Available: citeseer.ist.psu.edu/goldberg02berkmin.html

[8] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient

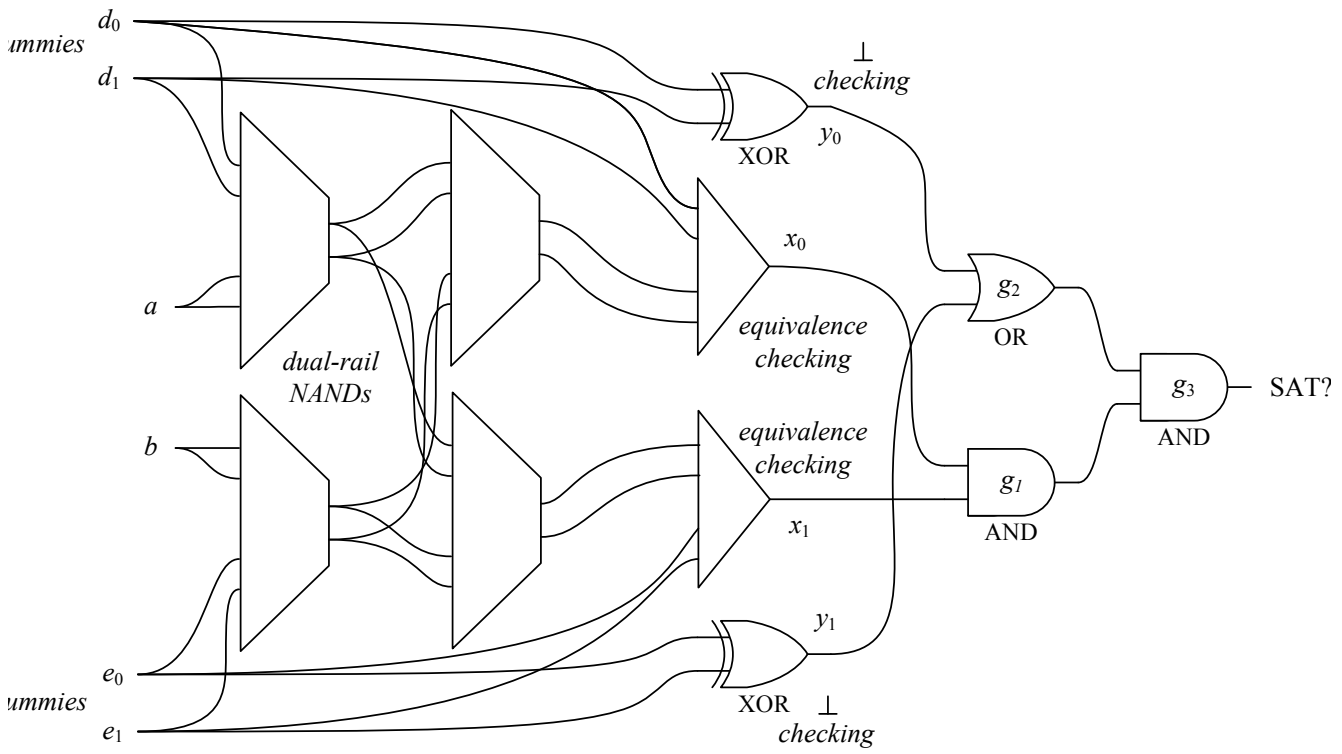

Fig. 8. The SAT circuit corresponding to the cyclic circuit in Figure 6.

SAT Solver," in *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001. [Online]. Available: citeseer.ist.psu.edu/moskewicz01chaff.html

[9] D. A. Huffman, "Combinational circuits with feedback," *Recent Developments in Switching Theory*, pp. 27 – 55, 1971.

[10] W. Kautz, "The necessity of closed circuit loops in minimal combinational circuits," *Computers, IEEE Transactions on*, vol. C-19, no. 2, pp. 162–164, Feb. 1970.

[11] R. Short, "A theory of relations between sequential and combinational realizations of switching functions," Ph.D. dissertation, 1961.

[12] C. McCaw, "Loops in directed combinational switching networks," Ph.D. dissertation, 1963.

[13] R. L. Rivest, "The necessity of feedback in minimal monotone combinational circuits," *IEEE Trans. Comput.*, vol. 26, no. 6, pp. 606–607, 1977.

[14] L. Stok, "False loops through resource sharing," in *ICCAD '92: Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, pp. 345–348.

[15] S. Malik, "Analysis of cyclic combinational circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, no. 7, pp. 950–956, Jul 1994.

[16] S. A. Edwards, "Making cyclic circuits acyclic," in *DAC '03: Proceedings of the 40th conference on Design automation*. New York, NY, USA: ACM, 2003, pp. 159–162.

[17] T. R. Shiple, "Formal analysis of synchronous circuits," Ph.D. dissertation, 1996, chair-Alberto Sangiovanni-Vincentelli.

[18] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Tech. Rep., 1992. [Online]. Available: citeseer.ist.psu.edu/sentovich92sis.html

[19] M. Riedel and J. Bruck, "Cyclic combinational circuits: Analysis for synthesis," 2003. [Online]. Available: citeseer.ist.psu.edu/riedel03cyclic.html

[20] M. D. Riedel and J. Bruck, "Timing analysis of cyclic combinational circuits." [Online]. Available: citeseer.ist.psu.edu/694687.html

[21] M. Yoeli and S. Rinon, "Application of ternary algebra to the study of static hazards," *J. ACM*, vol. 11, no. 1, pp. 84–97, 1964.

[22] R. Bryant, "Boolean analysis of mos circuits," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 6, no. 4, pp. 634–649, July 1987.

[23] J. Brzozowski and C.-J. Seger, "Asynchronous circuits," *Springer-Verlag*, 1995.

[24] M. Mendler and M. Fairtlough, "Ternary simulation: A refinement of binary functions or an abstraction of real-time behaviour," 1996. [Online]. Available: citeseer.ist.psu.edu/article/mendler96ternary.html

[25] E. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM Journal of Research and Development*, vol. 9, pp. 90–99, 1965.

[26] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1,

no. 2, pp. 146–160, 1972. [Online]. Available: http://link.aip.org/link/?SMJ/1/146/1

[27] M. Ganai, L. Zhang, P. Ashar, A. Gupta, and S. Malik, "Combining strengths of circuit-based and cnf-based algorithms for a high-performance sat solver," 2002. [Online]. Available: citeseer.ist.psu.edu/ganai02combining.html

[28] A. M. et al., "Abc: A system for sequential synthesis and verification." [Online]. Available: http://www.eecs.berkeley.edu/$\sim$alanmi/abc/

[29] N. S. et al., "Minisat v1.13 - a sat solver with conflict-clause minimization," Tech. Rep.