Analysis of Cyclic Combinational Circuits

Sharad Malik (sharad@ee.princeton.edu)
Dept. of Electrical Eng., Princeton Univ.

Abstract

A logic circuit is said to be combinational if the function it computes depends only on the inputs applied to the circuit, and is sequential if it depends on some past history in addition to the current inputs. Circuits that have an underlying topology that is acyclic are combinational, since feedback is a necessary condition for it to be sequential. However, it is not a sufficient condition since there exist combinational logic circuits that are cyclic. These occur often in bus structures in data paths. Traditional formal techniques in logic synthesis, logic analysis and timing analysis of combinational circuits have restricted themselves to acyclic combinational circuits since they have been unable to handle the analysis of circuits with cycles. Thus, in practice, these circuits are handled using clumsy work-arounds, which is obviously undesirable. This paper presents a formal analysis of these circuits, and presents techniques for the logical and timing analysis of such circuits. These techniques are practically feasible on reasonably large circuits encountered in practice.

1 Introduction

Even though the phrase *cyclic combinational circuits* may seem an oxymoron, these circuits do infact exist. For example, the circuit in Figure 1 is combinational since z is equal to the present value of x, and does not depend on the past history of inputs. This can be easily verified by checking the circuit output for both $x = 0$ and $x = 1$. While in this circuit there exists a logical redundancy, and the circuit can be reduced to an acyclic form by the removal of this redundancy, this in not true in general. As early as in 1970, Kautz demonstrated the existence of circuits for which he was able to prove that the minimal form must have cycles [8].

The existence of these circuits is not restricted to the world of abstract researchers, they occur often in practice. Stok has pointed out their existence in circuits synthesized from high level descriptions [12]. They also occur often in circuits with bus structures. Both these families of instances are motivated by the following abstract example. Consider the function:

```
z = if(c) then F(G(x)) else G(F(x))
```

Here c is some logical condition, and x is an input argument (possibly a vector of Boolean variables). Any acyclic implementation of this function must have two instances of at least one of F or G, since F follows G in the then part, and the converse is true for the else part. However, Figure 2 shows a cyclic implementation of this function that uses only one instance of both F and G, exploiting the fact that for any input assignment to c only one of the two possibilities, F following G, or G following F will happen.

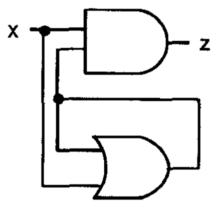


Figure 1: Cyclic Combinational Circuits: A Simple Example

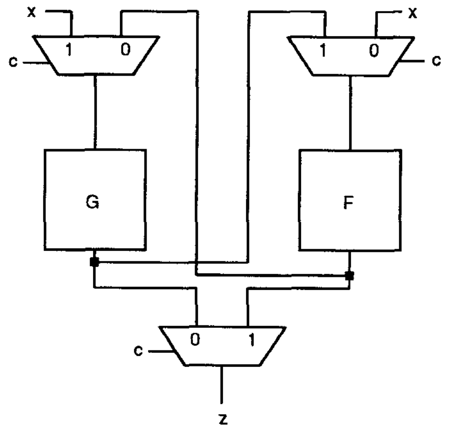


Figure 2: Cyclic Combinational Circuits: Abstract Example

This abstract example can be transformed to a practical example by replacing F and G by a shifter and an adder respectively and considering the function:

```
z = if(c) then shift(add(a, b), d)
         else add(shift(a, d), b)
```

Here a and b are the data arguments, and d is the amount the shifter must shift its first argument by. Depending on the logical condition c, the shift is either performed on input a before addition to b or the

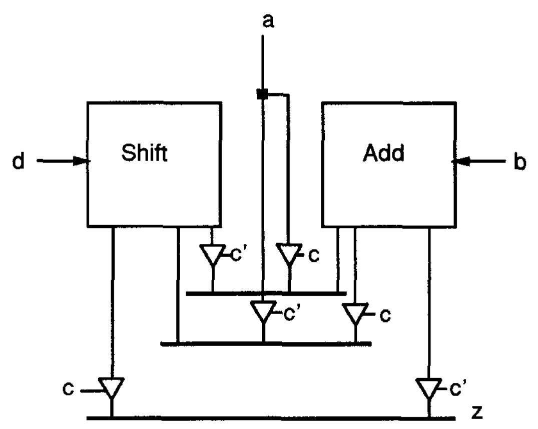


Figure 3: Cyclic Combinational Circuits: Practical Example

result of adding a and b is shifted after addition. The cyclic version of the circuit implementation is shown in Figure 3. It is common to use tri-state buses instead of multiplexors in data paths, however the two are logically equivalent.

Traditional formal techniques in logic synthesis, logic analysis and timing analysis of combinational circuits have restricted themselves to acyclic combinational circuits since they have been unable to handle the analysis of circuits with cycles. Stok laments the lack of existence of logic and timing analysis programs that handle cycles in combinational logic circuits. This motivates his work where he proposes a technique for resource allocation in high level synthesis that avoids the creation of such cycles [12]. In addition to requiring special attention, this possibly results in non-optimality in the resources needed. Inability to handle cycles has led to some logic synthesis programs prohibiting cycles from ever existing in the circuits they handle [2]. However, designers do design cyclic combinational circuits, and in practice these are handled using clumsy work-arounds, which is obviously undesirable. These work-arounds typically involve a lot of special casing and even specific solutions for different circuits. This paper presents a formal analysis of these circuits, and presents techniques for the logical and timing analysis of such circuits. These techniques are practically feasible on reasonably large circuits encountered in practice.

2 Preliminaries

A *combinational logic circuit* is described by specifying a set of gates and the interconnections between them. For purpose of this paper, we will consider only the *simple gates*, NOT, AND, OR. However, this is just for ease of exposition and is not a limitation of the analysis techniques presented later in the paper.

A *controlling value* on a gate input is one that determines the output of a gate independent of the values on the other inputs. For example, 0 is a controlling value for an AND gate and 1 for an OR gate. A *non-controlling value* on a gate input is one that does not determine the output of a gate independent of the values on the other inputs. For example, 0 is a non-controlling value for an OR gate and 1 for an AND gate.

A *path* in a circuit is an alternating sequence of gates and connections,
$\{g_0, c_0, ..., g_n, c_n, g_{n+1}\}$, where connection c_i, $1 \leq i \leq n$, connects the output of gate g_i to an input of gate g_{i+1}. g_0 is a primary input and g_{n+1} is a primary output. With each gate g, we associate a delay $d(g)$.

A path is said to be *simple* if each gate in the circuit appears in the path no more than once.

The *length* of a path $P = \{g_0, c_0, ..., g_n, c_n, g_{n+1}\}$, is defined as $length(P) = \sum_{i=0}^{n+1} d(g_i)$.

An *event* is a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ at a gate. Consider a sequence of events, $\{r_0, r_1, ..., r_n\}$ occurring at gates $\{g_0, g_1, ..., g_n\}$ along a path, such that r_i occurs as a result of event r_{i-1}. The event r_0 is said to propagate along the path.

A path is said to be *true* if it is can propagate an event for some input stimulus.

The symbol $\cap$ is used to denote the set intersection or the logical AND operation depending on the context. Similarly the symbol $\cup$ is used to denote the set union or the logical OR operation.

3 Functional Analysis

3.1 Intuitive Reasoning

It is possible that a circuit may be combinational under the assignment of specific delay values to the circuit components (gates and wires), but may be sequential otherwise. It is natural to expect the temporal behavior of a circuit to be a function of the delay values, but it is undesirable for its functional or logical behavior to be dependent on the delay values. Thus, when we classify a circuit as being combinational, we would like it to be combinational regardless of the delay values on circuit components. Indeed, logic designers designing cyclic circuits do not take into account delays during the logical design of these circuits; we did not need to know what the delay values were to classify the circuits in Figures 2 and 3 as being combinational. It is this delay independent notion of combinational circuits that we will use in the rest of the paper.

Cyclic combinational circuits have structural feedback, however there is no logical feedback that is transmitted to the primary outputs. To understand this further, let us examine the simple circuit shown in Figure 4(a). (This is the same as shown earlier in Figure 1.) When $x = 0$, there is logical feedback in the circuit, since y depends on the previous value of y. However, for this value of x, this feedback is not transmitted to the primary output z, since z is equal to 0 independent of the value of y. Thus, we see that the primary outputs may be combinational even when some intermediate signals in the circuit are sequential. In fact, parts of the circuit may even display oscilla-

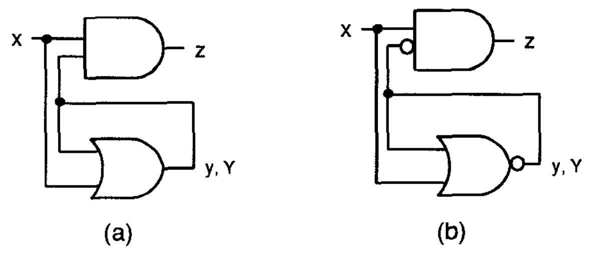


Figure 4: Cyclic Combinational Circuits with Sequential Parts

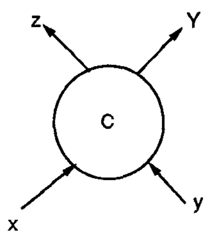
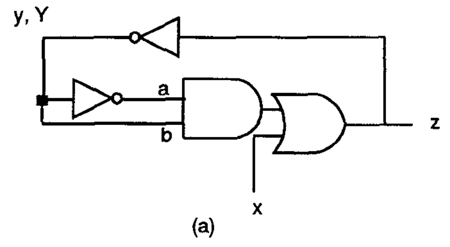


Figure 5: Breaking a Single Feedback Connection



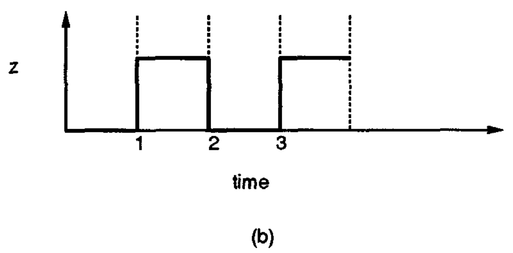


Figure 6: A Seemingly Combinational Circuit

tory behavior. This is seen in Figure 4(b) where y oscillates when $x = 0$ even though z is combinational.

To explore this further, let us consider cyclic circuits where breaking a single connection breaks all cycles. Let us call the new input and output of the acyclic circuit thus created as y and Y as shown in Figure 5. Here x and z denote the vector of primary inputs and outputs of the circuit. x_i and z_i will be used to denote the i^{th} primary input and output, respectively.

Let $D(Y, y)$ be the set of assignments to x for which the value Y depends on the value of y. Similarly, let $D(z, y)$ be the set of assignments to x for which z depends on y. The term ***depends*** is being used in an intuitive way here, a stricter definition of it and a description of how the set D is computed is deferred till later in the paper. There is logical feedback at signal y as long as $D(Y, y)$ is not empty. If z is to be combinational, then $D(z, y)$ must be disjoint from $D(Y, y)$, i.e.

$$D(z, y) \cap D(Y, y) = \phi \tag{1}$$

If this were not so, then for any member of the non-empty intersection, the value on the signal y in the original circuit would depend on its previous value, i.e. it would be sequential, and z would depend on this sequential value, making it sequential.

Thus we see that Equation 1 represents a necessary condition for the circuit to be combinational. It is easy to see that it is sufficient too; if it holds then there exist no assignments to x for which y is sequential and at the same time z depends on y.

Let us now try and define the function D more precisely. Traditionally the Boolean difference of z with respect to y, denoted by $\frac{\partial z}{\partial y}$, has been used to describe the set of conditions (assignments to other variables that z depends on) for which z depends on the value of y. For any assignment in $\frac{\partial z}{\partial y}$, toggling y would result in z toggling. Let us tentatively use this definition for $D(z, y)$, i.e. let $D(z, y) \equiv \frac{\partial z}{\partial y}$. For both the circuits in Figure 4 we have the following:

$$\frac{\partial z}{\partial y} = x \qquad \frac{\partial Y}{\partial y} = \overline{x}$$

Thus, $\frac{\partial z}{\partial y} \cap \frac{\partial Y}{\partial y} = \phi$, and, as expected, the circuits will be classified as being combinational. Let us now consider the example shown in Figure 6(a). At first glance it appears that z must be combinational since the output of the AND gate must always be 0 and hence $z = x$. This, however, is incorrect. Let the inverters in the figure have a delay of 1 time unit each and the other gates have zero delay. Further, assume that when the circuit is powered on, the signals a and b both have the logical value 0 on them and $x = 0$. With this assumption on the starting values and the given delay values on the gates in the circuit, it is easy to see that the circuit output z changes as shown in Figure 6(b) and will oscillate indefinitely. z is far from being combinational! But $\frac{\partial z}{\partial y} = 0$ when the connection y is broken to make the circuit acyclic; so z should not depend on y. What is wrong with using the Boolean difference here is that it assumes signals in the circuit to reach stable values. That is obviously true only for those signals whose values are uniquely set by the x variables. Nothing can be said for the other signals. How do we capture this condition?

gate type	$f_g.0$	$f_g.1$
NOT	$f_{g_1}.1$	$f_{g_1}.0$
AND	$\cup_i f_{g_i}.0$	$\cap_i f_{g_i}.1$
OR	$\cap_i f_{g_i}.0$	$\cup_i f_{g_i}.1$

Table 1: Calculus for Ternary Symbolic Simulation

3.2 Ternary Symbolic Simulation

A brief digression into ternary symbolic simulation is needed here. Ternary symbolic simulation was introduced by Bryant in the analysis of switch level circuits [4]. Ternary valued logic permits functions/signals to have a third value X (unknown) in addition to the two values 0 and 1 used in switching functions. For any assignment to the inputs x, each signal in the circuit evaluates to one of 1, 0 or X. Thus for each signal the input space (assignments to the inputs) is partitioned into three parts, one for each of the three possible values. Let s be a signal in the circuit and let $f_s.1(x)$ and $f_s.0(x)$ be switching functions defined as follows. $f_s.1(x)$ evaluates to 1 for exactly those assignments to x for which s evaluates to a 1. Similarly, $f_s.0(x)$ evaluates to a 1 for exactly those values of x for which s evaluates to 0. These two functions are sufficient to represent the ternary valued function being computed at s since input assignments for which s evaluates to X are the remaining ones, i.e. those for which $\overline{(f_s.1(x) \cup f_s.0(x))}$ holds true. Table 1 shows how the outputs of simple gates are evaluated using this "dual rail encoding" for ternary functions, given the functions for the inputs of the gates. Here g is the output of the gate and g_i are the inputs of the gate, with i varying over the inputs.

3.3 Logical Analysis

Armed with the machinery of ternary symbolic simulation, let us continue our analysis of cyclic combinational circuits. As before, for now let us restrict ourselves to circuits of the form shown in Figure 5; for these circuits breaking a single connection, y, breaks all cycles in the circuit. The ternary valued functions for the inputs to this acyclic circuit are as follows. For y: $f_y.0 = 0$, $f_y.1 = 0$, for a primary input x_i: $f_{x_i}.0 = \overline{x_i}$, $f_{x_i}.1 = x_i$. This captures the fact that the values for the primary inputs are completely known while those for the feedback signal y are completely unknown. With the functions available for the primary inputs, the calculus of Table 1 can be used with a depth first traversal of the acyclic circuit graph to evaluate the functions at Y and z. Then, $\overline{(f_Y.1(x) \cup f_Y.0(x))}$ denotes those assignments of x for which the value of Y cannot be determined to be a logical 1 or a 0 using just the values of x, i.e. these are the assignments for which Y depends on y. Similarly, $\cup_i\overline{(f_{z_i}.1(x) \cup f_{z_i}.0(x))}$ denotes the set of assignments for which at least one primary output (i varies over the primary outputs) depends on y. This mathematical definition of the intuitive notion "depends" captures the fact that a signal s depends on y as long as the assignment to the x variables does not by itself determine the logical value 0 or 1 of s. *The ternary calculus makes sure that no assumptions are made about stable values on any signals except for the primary input variables x.*

Using this definition for D in Equation 1 we see that following must be true for the circuit to be combinational.

$$(\cup_i\overline{(f_{z_i}.1(x) \cup f_{z_i}.0(x))}) \cap \overline{(f_Y.1(x) \cup f_Y.0(x))} = 0 \tag{2}$$

For the example shown in Figure 6, we see that:

$$f_z.0 = 0 \qquad f_z.1 = x$$

$$f_Y.0 = x \qquad f_Y.1 = 0$$

Thus, we see that Equation 2 is not satisfied. In particular the assignment to x for which both z and Y depend on y is $x = 0$, which is the case demonstrated in Figure 6(b).

3.4 Sensitization of Non-Simple Paths

Let us see what Equation 2 means in terms of paths in the circuit. Consider a path, as shown in Figure 7, from the primary inputs to the primary outputs in the circuit, that is not simple, i.e. it contains a cycle. (Note that not all gates along the path have been shown in this figure.) At least one of the following two things must happen for this path for each assignment of the primary inputs if the circuit is combinational.

- There must be some gate in the cycle which has an off-path input that is at a stable controlling value. This is indicated by the OR gate with a side input set at 1 in the figure. This side input determines the output of the gate and thus effectively cuts off information from circulating in the cycle, i.e. it logically breaks the cycle.
- There must be some gate on the part of the path from the cycle to the primary output that has an off-path input with a stable controlling value. This is indicated by the AND gate with a side input set at 0 in the figure. This side input determines the output of the gate and thus effectively cuts off information from the cycle from reaching the primary output.

3.5 Iterative Function Computation

The discussion in the previous sections describes the condition that we need to check in order to classify the circuit as being combinational or not. However, it does not provide a method for computing the switching function being computed by the circuit. Interestingly, ternary symbolic simulation can be used to even compute this function. This is done as follows. Once $f_Y.1$ and $f_Y.0$ have been computed, these values are used for $f_y.1$ and $f_y.0$ in another iteration of the ternary symbolic simulation of the acyclic circuit. This captures the fact that the function at signal

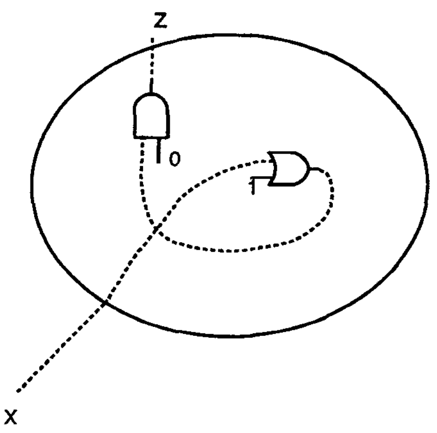


Figure 7: A Non-Simple Path

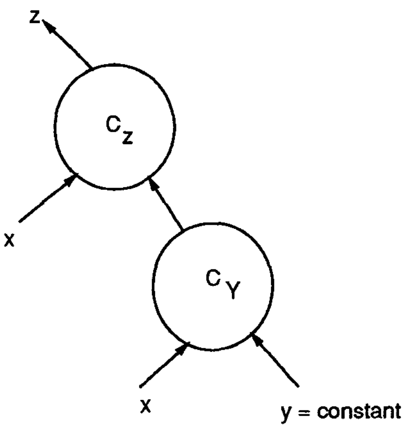


Figure 8: A Functionally Equivalent Acyclic Circuit

Y will be fed back to the acyclic circuit at y. Since we will be computing the functions at signals at several iterative steps, we will indicate this using a superscript on the function name. The absence of a value in the superscript indicates the initial or the zero-th iteration, which is what we have seen thus far. Thus, $f_y^i.1 = f_Y^{i-1}.1$ and $f_y^i.0 = f_Y^{i-1}.0$. The functions for the primary input variables x are invariant over the iteration number, i.e. $f_x^i.1 = x$ for all i, similarly $f_x^i.0 = \overline{x}$ for all i.

A direct consequence of Equation 2 is that

$$\cup_i \overline{(f_{z_i}^1.1(x) \cup f_{z_i}^1.0(x))} = 0 \qquad (3)$$

That is, for each input assignment of x, each of the primary outputs z_i are uniquely determined to be either 1 or 0 after the first iteration. Thus, the switching function being computed at primary output z_i is $f_{z_i}^1.1$ which is the same as $\overline{f_{z_i}^1.0}$. To see why this is so, let $\vec{x}$ be a particular input assignment and let z_i be some output. The following two cases are possible for the value of z_i for input $\vec{x}$:

Case 1: $f_{z_i}^0.0(\vec{x}) \cup f_{z_i}^0.1(\vec{x}) = 1$
In this case the value of z_i is determined to be 1 or 0 after the initial iteration.

Case 2: $f_{z_i}^0.0(\vec{x}) \cup f_{z_i}^0.1(\vec{x}) = 0$
In this case the value of z_i is not determined to be 1 or 0 after the initial iteration. In this case, z_i depends on the value of y. However, from Equation 2 we know that in this case $f_Y^0.0(\vec{x}) \cup f_Y^0.1(\vec{x}) = 1$, i.e. the value of Y is uniquely determined to be a 1 or 0 since it cannot depend on y for this input assignment. Since $f_y^1.0(\vec{x}) = f_Y^0.0(\vec{x})$ and $f_y^1.1(\vec{x}) = f_Y^0.1(\vec{x})$ $f_{z_i}^1.0(\vec{x}) \cup f_{z_i}^1.1(\vec{x}) = 1$, i.e. z_i is uniquely determined at the end of the first iteration.

The correctness of Equation 3 follows from the above description.

The iterative function computation also directly gives us an acyclic circuit that is functionally equivalent to the cyclic circuit that we started with. This is shown in Figure 8. Here, C_z is the circuit computing z and C_Y is the circuit computing Y. In C_Y we can set y to any constant value (0 or 1) since z does not depend on it.

We are now in a position to relax the condition that we have imposed thus far on the circuit, i.e. breaking a single connection breaks all the cycles in the circuit. While that condition made it easy for us to gain some insight into the nature of the problem, it is obviously restrictive, and at this point no longer necessary. Let the cardinality of the feedback arc set be k, i.e. breaking k connections breaks all the cycles in the circuit. Let the additional inputs and outputs created by breaking these connections be $y_1, y_2, \ldots, y_k$ and $Y_1, Y_2, \ldots, Y_k$ respectively. For the original cyclic circuit to be combinational, the following is both necessary and sufficient.

$$\cup_i \overline{(f_{z_i}^j.1(x) \cup f_{z_i}^j.0(x))} = 0 \quad \text{for some} \quad j \leq k \qquad (4)$$

Thus, in no more than k iterations the functions at the primary outputs must not depend on the feedback variables. The derivation of Equation 4 follows the same lines as that of Equation 3. Intuitively the k iterations permit the primary input to determine the value at the primary outputs through a path that traverses all the feedback connections as sketched in Figure 9. All paths in the circuit will be considered in k iterations. Thus, there is no need to consider any more iterations that that. If in k iterations the values of all output signals are known to be 1 or 0 for each input assignment, then the circuit is combinational, else it is sequential. Note that if the circuit is sequential, the LHS of Equation 4 provides the set of input assignments under which the circuit displays sequential behavior. If all of these are known to be don't cares, then for the care assignments the circuit is still combinational.

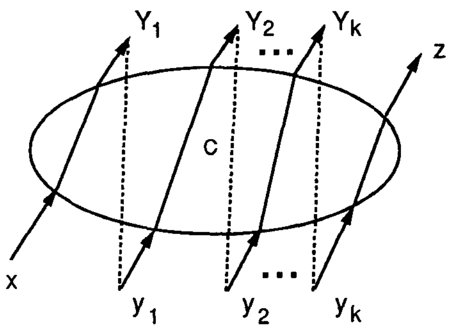


Figure 9: A k-iteration Path

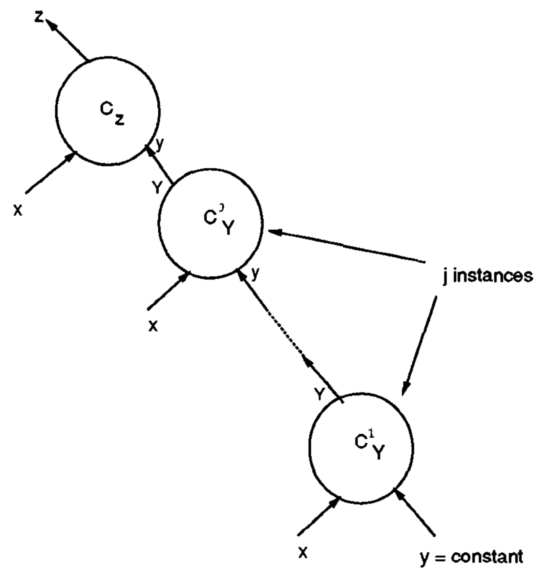


Figure 10: Equivalent Acyclic Circuit: General Case

Correspondingly, the equivalent acyclic version of the circuit is shown in Figure 10. Here j copies of C_Y are needed, where j is the smallest iteration count for which Equation 4 holds. The superscript on C_Y serves as an index for the copy. In C^l_Y only those Y signals need be computed which stabilize (value known to be 1 or 0 for all input assignments) in less than l iterations.

It is interesting to note that j may be much smaller that k. In particular, for the circuit in Figure 3, k is the width of the datapath, but $j = 1$ independent of k. Thus that circuit is a one-iteration cyclic combinational circuit.

Based on our present insight into cyclic combinational circuits, we believe that most practical instances of these will be one-iteration cases.

From the earlier discussion using Figure 7 it is easy to see that any feedback arc set will suffice for purposes of this algorithm. All we are checking for here is that each non-simple path is blocked either in the cycle or in the part from the cycle to the primary output by means of an off-path input that is at a stable controlling value. Determining a feedback arc set is linear in the size of the circuit and can be done using a depth first search starting at the primary inputs.

4 Complexity Analysis

Consider the following two related problems that are being examined in this paper.

P1: Is a given cyclic circuit combinational for all assignments of delays to the gates?

P2: Is a given cyclic circuit sequential for some assignment of delays to the gates?

P1 and P2 are clearly complementary problems and it is sufficient to examine the complexity of any one of them. For a circuit to be sequential we need to determine an input assignment for which the circuit will demonstrate sequential behavior. This looks similar to the satisfiability problem for Boolean functions, and it would seem relatively straightforward to show that P2 is in NP. However, this is not easy. The reason for this is that even though we can "guess" this input assignment, there is no known polynomial time algorithm that will simulate the circuit and tell us if the output is sequential for some delay assignment. We cannot guess the delay assignment since delay values are real numbers. Thus, it is not easy to demonstrate that P2 in in NP.

Let us now go back to Equation 4. Finding a satisfying input assignment for the Boolean function on the LHS of the equation is equivalent to detecting that it is sequential. Thus, we can now "guess" an input assignment for the original cyclic circuit, and check if it satisfies Equation 4 in time that is polynomial in the circuit size. Thus, a corollary of Equation 4 is that P2 is in NP.

Having established that P2 is in NP, it is now easy to show that it is in fact NP-complete by reducing satisfiability to it. Let f be any Boolean function that we wish to check for satisfiability. Let z be computed by the following equation corresponding to a single AND gate.

$$z = AND(z, f)$$

Clearly z is sequential if and only if f has a satisfying assignment. In fact it is precisely for the satisfying assignments of f that z is sequential. This completes the proof that P2 is NP-complete. Since P1 is the complementary problem for P2, P1 is co-NP-complete.

This precise classification of these problems is important since it highlights the intrinsic nature of these problems. The lack of a classification of these problems may have been responsible for a lack of a clean solution to them in the past.

5 Timing Analysis

Section 3 describes techniques to check if a cyclic circuit is combinational, and determine the switching function being computed by the circuit if it indeed is combinational. However, it does not tell us much about the temporal behavior of the circuit. If this circuit is to be used in a synchronous design, then the maximum time taken for the circuit outputs to stabilize after the inputs have been applied will determine the period of the clock that controls the memory elements. How do we determine this period?

In acyclic combinational circuits, this is relatively easy since the length of the longest path in the circuit provides an upper bound on the period. This bound is tight if there exists an input stimulus that will actually sensitize a path of this length. If no such stimulus exists then all paths of this length are said to be false and in this case the bound is not tight. Functional timing analysis techniques developed in recent years (e.g. [10, 5, 11, 6, 1]) have been successful in determining a tighter bound by taking into account the functionality of the circuit elements and thus eliminating the false paths during the analysis.

This section examines the problem of determining the maximum delay of a cyclic combinational circuit and provides several different techniques of computing an upper bound on the circuit delay. Of these, only one is guaranteed to be tight. In the following discussion, we assume that the cyclic circuit, C, under consideration, has already been determined to be combinational.

Let us examine the first of these bounds. From the discussion accompanying Figure 7, we know that no non-simple path from a primary input to a primary output in the circuit is true. Each non-simple path is blocked for each primary input assignment in at least one of the following two parts:

- the cycle
- sub-path from the cycle to the primary output

A path is said to be *blocked* at a gate for some primary input assignment if there is an off-path input to that gate with a controlling value, thus blocking propagation of any event along the path beyond this gate. Thus, a simple bound on the delay of the circuit is the length of the longest simple path. Let us call this bound the $\mathcal{L}_1$ bound. This bound is analogous to the longest path bound for acyclic circuits. In the case of acyclic circuits, this was easy to compute, in the cyclic case, finding the length of the longest simple path is NP-hard. While there has been significant practical success in finding efficient heuristics for other NP-hard problems in design automation, no successful heuristics have been reported for this problem. A related result in theoretical computer science points to the difficulty of even approximating the length of the longest path by showing that this problem is NP-hard, even for bounded degree graphs [7]. This leaves us with the interesting and challenging proposition of finding efficient heuristics for the longest simple path problem. We are currently examining this issue. The $\mathcal{L}_1$ bound is not guaranteed to be tight, since the longest simple path may not be true, there may exist no input stimulus that sensitizes it. Thus, we need to determine the length of the longest true simple path.

Computing the length of the longest true simple path seems unmanageable. However, the equivalent acyclic combinational circuit (subsequently referred to as A_1) shown in Figure 10 comes in handy here. It is interesting to see the relationship between the sets of paths in this acyclic circuit, $P(A_1)$, and the set of simple paths in the original cyclic circuit, $P(C)$. $P(A_1)$ is neither contained in, nor does it contain $P(C)$. To see why this is so, $P(A_1)$ may have paths that correspond to non-simple paths in the cyclic circuit. These are the paths that traverse two copies of the same gate in the acyclic version of the circuit. On the other hand $P(A_1)$ may not have all the simple paths. Recall that the iterative computation described in the previous section stopped after j iterations when the outputs stabilized. Thus, while $P(A_1)$ is guaranteed to contain, for each input assignment, one path such that this path gets the stable final value to the output, it may not contain all the simple paths. However, the length of the longest path in $P(A_1)$ serves as a bound for the delay since all outputs do stabilize to their final value no later than this length. Let us call this bound the $\mathcal{L}_2$ bound. However, the longest path in $P(A_1)$ may not be true and hence this bound is not tight. Thus, a tighter bound is the length of the longest true path in $P(A_1)$, let us denote this as $\mathcal{L}_3$. Even this bound is not tight. This follows from the following observation. For each input assignment for which the longest paths in $P(A_1)$ get the stable values to the outputs, there may a shorter simple path in $P(C)$ that is not contained in $P(A_1)$ that gets that value to the output faster. Thus, the longest true paths in $P(A_1)$ may still be false in $P(C)$.

The longest true paths in $P(C)$ can actually be found using functional timing analysis in an acyclic circuit. This acyclic circuit is similar to the one shown in Figure 10, except that j is now replaced by k, (subsequently referred to as A_2). This guarantees that A_2 contains all the simple paths of C. Let $P(A_2)$ be the set of paths in this circuit. Now the longest true path in $P(A_2)$ is the longest true simple path in $P(C)$. This bound, termed $\mathcal{L}_4$ is a tight bound on the circuit delay. Figure 11 summarizes the relationship between these bounds. The only unknown relationship is the one between $\mathcal{L}_1$ and $\mathcal{L}_2$.

We feel that in practice, $\mathcal{L}_1$ and $\mathcal{L}_3$, will be tight bounds, while $\mathcal{L}_2$ will be loose. Of these, we have established techniques for computing $\mathcal{L}_3$. As mentioned earlier, it will be interesting to see how these compare in computation time to those that will be developed for $\mathcal{L}_1$.

For the circuit in Figure 3, $\mathcal{L}_1$ and $\mathcal{L}_3$ both provide the tight bound of the delay through three multiplexors and one adder and one shifter. $\mathcal{L}_2$, however, returns the delay of five multiplexors and two adders and two shifters.

$\mathcal{L}_1$	longest simple path in C
$\mathcal{L}_2$	longest path in A_1
$\mathcal{L}_3$	longest true path in A_1
$\mathcal{L}_4$	longest true path in A_2

	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_3$	$\mathcal{L}_4$
$\mathcal{L}_1$	$=$	?	$\geq$	$\geq$
$\mathcal{L}_2$		$=$	$\geq$	$\geq$
$\mathcal{L}_3$			$=$	$\geq$
$\mathcal{L}_4$				$=$

Figure 11: Alternate Bounds for Circuit Delay

Width in bits	Logical Analysis	Timing Analysis
8	0.2	1.9
16	0.7	10.4
32	1.9	132.2

All times are in seconds on a Sun 4/75GX.

Table 2: Experimental Results for Different Datapath Widths

6 Implementation and Experiments

The iterative function computation using the dual-rail encoding for ternary functions is very easy to implement using a Binary Decision Diagram representation [3] for representing $f_s^i.1$ and $f_s^i.0$.

For delay bounds, we are currently examining heuristics for computing the longest simple path in cyclic circuits (the $\mathcal{L}_1$ bound). For the $\mathcal{L}_3$ bound, we are using the functional timing analysis program presented in [1].

Very few examples are available for use as benchmark circuits for this purpose. Table 2 shows computation time for the logical analysis using iterative computation, as well as for computing the $\mathcal{L}_3$ bound for different widths of the datapath circuit shown in Figure 3 with a one-bit shift.

7 Current Related Work

This paper describes the analysis of cyclic combinational circuits. Since we know that cyclic combinational circuits may be potentially smaller that their acyclic versions, can we use any of the ideas developed in the analysis for logic synthesis of cyclic combinational circuits? In particular, can we incorporate this in a general logic synthesis environment, thus removing the restriction that the circuits generated by them need to be cyclic. Some preliminary ideas in that direction have been developed and will be reported shortly [9].

8 Acknowledgments

Thanks to Richard Rudell for pointing out the practical interest in this problem, and also for supplying the examples shown in Figures 2 and 3. Thanks to Ajai Kapoor for stimulating discussions on complexity issues.

This research is supported by an NSF Research Initiation Award and an IBM Faculty Development Award.

References

[1] P. Ashar, S. Malik, and S. Rothweiler. Functional timing analysis using ATPG. In *Proceedings of the European Design Automation Conference*, February 1993.

[2] R. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang. MIS: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design*, pages 1062–1081, November 1987.

[3] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. In *IEEE Transactions on Computers*, volume C-35, pages 677–691, August 1986.

[4] R. E. Bryant. Boolean analysis of MOS circuits. *IEEE Transactions on Computer-Aided Design*, pages 634–649, July 1987.

[5] H. C. Chen and D. H. C. Du. Path sensitization in critical path problem. *IEEE Transactions on Computer-Aided Design*, pages 196–207, February 1993.

[6] S. Devadas, K. Keutzer, and S. Malik. Delay computation in combinational circuits: Theory and algorithms. In *Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[7] D. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. Technical report, Dept. of Computer Science, Stanford University, 1993.

[8] W. H. Kautz. The necessity of closed loops in minimal combinational circuits. *IEEE Transactions on Computers*, pages 162–164, February 1970.

[9] S. Malik and P. Ashar. Synthesis and testing of cyclic combinational circuits. Technical report, Dept. of Electrical Engineering, Princeton University, 1993. In Preparation.

[10] P. C. McGeer and R. K. Brayton. *Integrating functional and temporal domains in logic design.* Kluwer Academic Publishers, 1991.

[11] P. C. McGeer, A. Saldanha, R. K. Brayton, and A. Sangiovanni-Vincentelli. Timing analysis and delay-fault test generation using path recursive functions. In *Proceedings of the International Conference on Computer-Aided Design*, November 1991.

[12] L. Stok. False loops through resource sharing. In *Proceedings of the International Conference on Computer-Aided Design*, pages 345–348, November 1992.