

# Cyclic Combinational Circuits: Analysis for Synthesis

Marc D. Riedel and Jehoshua Bruck  
California Institute of Technology  
Mail Code 136-93, Pasadena, CA 91125  
E-mail: {riedel, bruck}@paradise.caltech.edu

*Abstract*— Digital circuits are called *combinational* if they are memoryless: they have outputs that depend only on the current values of the inputs. Combinational circuits are generally thought of as acyclic (i.e., feed-forward) structures. And yet, cyclic circuits can be combinational. In previous work, we showed that introducing cycles permits optimizations of area. We proposed a general methodology for the synthesis of multilevel networks with cyclic topologies and incorporated it in a logic synthesis environment. In trials, benchmark circuits were optimized significantly, with improvements of up to 30% in the area.

In this paper, we discuss the role of combinationality analysis in the context of synthesis. We present a symbolic framework for analysis based on a first-cut strategy. Unlike previous approaches, our method does not require ternary-valued simulation. It is formulated recursively, and thus it permits us to cache analysis results for common sub-networks through iterations of the synthesis process. We also discuss timing analysis of cyclic combinational circuits.

*Keywords*— Feedback, Logic Synthesis, Combinational Circuits

## I. INTRODUCTION

THE term *combinational* means that a circuit has outputs that depend only on the current values of the inputs (i.e., it is memoryless); the term *sequential* means that a circuit has outputs that may depend upon past as well as current input values (i.e., it has memory).

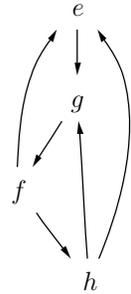
Combinational circuits are generally thought of as acyclic structures, and sequential circuits as cyclic structures. In fact, *combinational* and *sequential* are often defined in this way. A collection of logic gates connected in an acyclic (i.e., loop-free) topology is clearly combinational. Regardless of the initial values on the wires, once the values of the inputs are fixed, the signals propagate to the outputs. There is a clear correspondence between the electrical behavior of the circuit and the abstract notion of the boolean functions that it implements. The behavior of a circuit with feedback is generally more complicated. Such a circuit may exhibit timing-dependent behavior (as in the case of an R-S Latch), and it may be unstable (as in the case of an oscillator).

And yet, cyclic circuits can be combinational. Consider

This work is supported in part by the “Alpha Project” at the Center for Genomic Experimentation and Computation, a National Institutes of Health Center of Excellence in Genomic Sciences. The Alpha Project is supported by a grant from the National Human Genome Research Institute (Grant no. P50 HG02370).

the example shown in Figure 1, a lookup table for the first 16 digits of  $\pi$ . Given inputs  $a, b, c, d$ , specifying a number  $i$

	$d, c, b, a$	$\pi$	$h, g, f, e$
0	0 0 0 0	3	0 0 1 1
1	0 0 0 1	1	0 0 0 1
2	0 0 1 0	4	0 1 0 0
3	0 0 1 1	1	0 0 0 1
4	0 1 0 0	5	0 1 0 1
5	0 1 0 1	9	1 0 0 1
6	0 1 1 0	2	0 0 1 0
7	0 1 1 1	6	0 1 1 0
8	1 0 0 0	5	0 1 0 1
9	1 0 0 1	3	0 0 1 1
10	1 0 1 0	5	0 1 0 1
11	1 0 1 1	8	1 0 0 0
12	1 1 0 0	9	1 0 0 1
13	1 1 0 1	7	0 1 1 1
14	1 1 1 0	9	1 0 0 1
15	1 1 1 1	3	0 0 1 1



$$\begin{aligned}
 e &= \bar{f}(a\bar{h} + c) + d\bar{h} + \bar{b} \\
 f &= \bar{a}\bar{d}\bar{g} + a(\bar{b}d + bc) \\
 g &= \bar{a}b\bar{c} + \bar{h}(a\bar{e} + \bar{a}d + \bar{b}c) \\
 h &= \bar{f}(a(c + d) + cd)
 \end{aligned}$$

Fig. 1. Example: Lookup table for the digits of  $\pi$ .

between 0 and 15 (in binary), the network yields outputs,  $e, f, g, h$ , specifying the  $i$ -th digit of  $\pi$  (in binary). Each output is specified as a function of the input variables and the other output functions. As shown, the network contains cycles  $((e, g, f), (e, g, f, h), \text{ and } (f, h, g))$ . In spite of this, the network is combinational. For each combination of input values, the network produces the correct outputs, regardless of the initial state and independently of all timing assumptions. To see this, consider specific input values. For instance, with  $a = 0, b = 0, c = 0, d = 0$ , the network simplifies to that shown in Figure 2, yielding the correct value of  $e = 1, f = 1, g = 0, h = 0$  (the first digit of  $\pi$ , namely 3). With  $a = 1, b = 1, c = 1, d = 1$ , the network simplifies to that shown in Figure 3, yielding the correct value of  $e = 1, f = 1, g = 0, h = 0$  (the 16th digit of  $\pi$ , namely 3). The reader may verify that the network implements all the values in between 0000 and 1111 correctly.

Although the premise of cycles in combinational circuits has been established, combinational circuits are not designed with feedback in practice. Except for relatively simple cases of feedback at the level of functional units, no one has attempted the synthesis of circuits with feedback at the logic level.

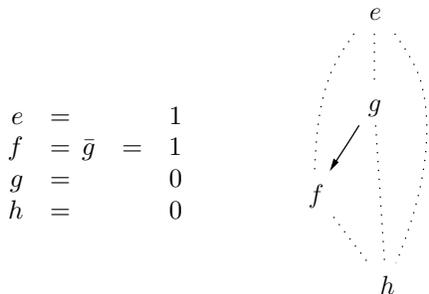


Fig. 2. Network in Figure 1 with  $a = 0, b = 0, c = 0, d = 0$ .

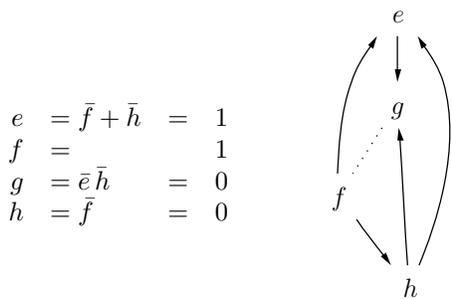


Fig. 3. Network in Figure 1 with  $a = 1, b = 1, c = 1, d = 1$ .

### A. Prior Work

In 1992, Stok observed that cycles sometimes occur in combinational circuits synthesized from high-level designs [16]. In such examples, feedback is carefully contrived, occurring when functional units are connected in a cyclic topology. Recently, Edwards pointed out that cycles arise in circuits synthesized from synchronous languages such as Esterel [7]. Most logic synthesis and verification tools balk when given cyclic designs for combinational logic. Stok’s solution to this dilemma is to disallow the creation of cycles in the resource-sharing phase of high-level synthesis; Edwards’ approach is to transform cyclic designs into equivalent acyclic ones.

In 1994, Malik addressed the issue of analyzing cyclic combinational circuits [8]. He proved that deciding whether a cyclic circuit is combinational or not is co-NP-complete, and he formulated an analysis algorithm for this task based on ternary-valued simulation. He also addressed the issues of timing analysis and testing for faults [9], [15]. In 1996, Shiple extended Malik’s work and set it on a firm theoretical footing [12]. He showed that the class of circuits that Malik’s procedure decides to be combinational are precisely those that are well-behaved electrically, according to the up-bounded inertial delay model [6]. He proposed refinements to Malik’s algorithm [13] and ex-

tended the concept to combinational logic embedded in sequential circuits [14].

### B. Contributions

In previous work, we showed that combinational circuits can be optimized significantly if cycles are introduced [10]. The intuition behind this is that, with feedback, all nodes can potentially benefit from work done elsewhere; without feedback, nodes at the top of the hierarchy must be constructed from scratch. We proposed a general methodology for the synthesis of multilevel networks with cyclic topologies and incorporated it in a general logic synthesis environment, namely the Berkeley SIS package [11]. Our approach is to optimize a multilevel description in the substitution phase, introducing feedback and potentially reducing the area. In trials with benchmark circuits, many were optimized significantly, with improvements of up to 30% in the cost (as measured by the literal count of the nodes expressed in factored form). In trials with randomly generated examples, very nearly *all* had cyclic solutions superior to acyclic forms. Thus, we argued for a paradigm shift in combinational circuit design: we should no longer think of combinational logic as acyclic in theory or in practice, since nearly all combinational circuits are best designed with cycles.

In this paper, we discuss the role of combinationality analysis – determining whether cyclic logic is combinational – in the context of synthesis. We present a symbolic framework for analysis based on a first-cut strategy. Unlike previous approaches, our method does not require ternary-valued simulation. It is formulated recursively, and thus it permits us to cache analysis results for common sub-networks through iterations of the search. We also discuss timing analysis of cyclic combinational circuits.

### C. Definitions and Notation

The exposition in this paper is based upon so-called symbolic operations, in which boolean functions are used to encode input patterns. The representation that we used in our implementation is based on Binary Decision Diagrams (BDDs) [5].

We use the standard notation: addition (+) denotes disjunction (OR), multiplication ( $\cdot$ ), denotes conjunction (AND), and an overbar ( $\bar{x}$ ) denotes negation (NOT). The **restriction** operation (also known as the cofactor) of a function  $f$  with respect to a variable  $x$ ,

$$f|_{x=v},$$

refers to the assignment of the constant value  $v \in \{0,1\}$  to  $x$ . The **composition** operation of a function  $f$  with respect to a variable  $x$  and a function  $g$ ,

$$f|_{x=g},$$

refers to the substitution of  $g$  for  $x$  in  $f$ . A function  $f$  **depends** upon a variable  $x$  iff  $f|_{x=0}$  is not identically equal

to  $f|_{x=1}$ . Call the variables that a function depends upon its **support set**.

The **universal quantification** operation (also known as consensus) yields a function

$$\forall (y_1, \dots, y_n) f$$

that equals 1 iff the given function  $f$  equals 1 for all  $2^n$  assignments of boolean values to the variables  $y_1, \dots, y_n$ . The **existential quantification** operation (also known as smoothing) yields a function

$$\exists (y_1, \dots, y_n) f$$

that equals 1 iff the given function  $f$  equals 1 for *some* assignment of boolean values to the variables  $y_1, \dots, y_n$ .

The **marginal** operation yields a function

$$f \downarrow (y_1, \dots, y_n)$$

that equals 1 iff the given function  $f$  is invariant for all  $2^n$  assignments of boolean values to  $y_1, \dots, y_n$ . For a single variable  $y$ , it equals 1 iff  $f|_{y=0}$  agrees with  $f|_{y=1}$ ,

$$f \downarrow y = f|_{y=0} \cdot f|_{y=1} + \overline{f|_{y=0}} \cdot \overline{f|_{y=1}}.$$

(For a single variable, the marginal is the complement of what is known as the boolean difference.) For several variables  $y_1, \dots, y_n$ , the marginal is computed as the universal quantification of the product of the marginals:

$$f \downarrow (y_1, \dots, y_n) = \forall y_1, \dots, y_n [(f \downarrow y_1) \cdots (f \downarrow y_n)].$$

(With several variables, the marginal is *not* the same as the complement of the boolean difference, in general.) For example, with

$$f = x_1 + x_2 y_1 + x_3 y_2 + x_4 y_1 y_2,$$

we have,

$$\begin{aligned} f \downarrow y_1 &= x_1 + x_3 y_2 + \bar{x}_2(\bar{x}_4 + \bar{y}_2), \\ f \downarrow y_2 &= x_1 + x_2 y_1 + \bar{x}_3(\bar{x}_4 + \bar{y}_1), \\ f \downarrow (y_1, y_2) &= x_1 + \bar{x}_2 \bar{x}_3 \bar{x}_4. \end{aligned}$$

Note that computing a marginal of  $n$  variables requires  $O(n)$  symbolic operations.

#### D. Network Model

Our model is at the level of abstraction applicable in the technology-independent phase of logic synthesis. Our goal is to construct a network that computes boolean functions of boolean input variables  $x_1, \dots, x_m$ . Internally, the network is specified as a collection of nodes  $\mathcal{N}$ . Associated with each node is a **node function**  $f_i$  and an **internal variable**  $y_i$ ,  $1 \leq i \leq n$ . The node functions depend on input variables as well as on internal variables. In the **dependency** graph, a directed edge is drawn from node  $i$  to node  $j$  iff the node function  $f_j$  associated with node  $j$  depends on the internal variable  $y_i$  associated with node  $i$ .

Also associated with each node is a **target function**  $g_i$  (in the case of acyclic networks this would be the “collapsed” function).<sup>1</sup>

The target functions depend on the input variables only. A subset of the nodes are designated as **output nodes**. For these, the target functions are the requisite output functions. If we substitute the target function  $g_j$  for each corresponding internal variable  $y_j$  in a node function  $f_i$ , we get the corresponding target function  $g_i$ ,

$$f_i|_{y_1=g_1, \dots, y_n=g_n} = g_i.$$

For a fixed assignment of inputs, call the network the **induced** network, and call the associated dependency graph the **induced** dependency graph. In the induced network, if a node function  $f_i$  doesn’t depend upon any internal variable (i.e., it evaluates to 0 or 1), then we may substitute this value for the corresponding internal variable  $y_i$  in other expressions. In this way, we can continue to simplify the network, until no further simplifications are possible. Call the result the **simplified induced** network.

#### E. Definition of Combinationality

A network is **combinational** iff it computes unique boolean output values for each boolean input assignment. We sometimes abuse this terminology and say that a network is combinational for a *specific* input assignment, meaning that it computes unique boolean output values for that input assignment. If there are “don’t care” conditions on the inputs, then it is sufficient if the network computes unique boolean values for input assignments in the “care” set.

This computation must hold:

- regardless of the initial state
- and independently of all timing assumptions.

**Proposition 1** *A network is combinational iff, for each assignment of boolean values to the inputs, all output nodes in the simplified induced network evaluate to definite boolean values.*

This definition of combinationality is functionally equivalent to that proposed in earlier work. Malik [8] suggested the ternary model for the analysis of cyclic combinational circuits. Following Bryant [4], his approach for deciding combinationality is based on ternary-valued simulation. He uses a “dual-rail” encoding (10 for one, 01 for zero, and 11 for “unknown”) to reduce the problem to boolean simulation.

<sup>1</sup>We use  $x_i, y_i, f_i, g_i$  when we refer to networks in the abstract. However, for the sake of readability, in our examples we use  $a, b, c, \dots$  for the input variables. We use  $e, f, g, \dots$  for the node functions, the internal variables, and the target functions: on the left-hand side of an equation the symbol refers to either a node function or a target function depending on the context; on the right-hand side, it refers to the associated internal variable.

## II. ANALYSIS

We formulate a symbolic framework for analysis that obviates the need for ternary-valued simulation. We tackle the problem with a divide-and-conquer approach: progressively smaller components of the network are analyzed for combinationality. We note that if a network's dependency graph can be divided into several distinct strongly connected components, then the analysis may be performed separately on each component. For simplicity, we assume that each node in the network is an output node.

### A. Symbolic Framework

We analyze which input assignments, when substituted into a node function, force it to a definite boolean value, independent of all internal variables in its support set. For a node function  $f_i$ , let  $I_i$  be the set of internal variables that it depends upon. The marginal operator, defined in Section I-C, is the required tool. If

$$f_i \downarrow I_i$$

holds, then  $f_i$  has a definite boolean value equal to the corresponding target function  $g_i$ . For a network  $\mathcal{N}$ , to obtain the restriction

$$\mathcal{N}|_{f_i},$$

the node  $f_i$  is removed, and the corresponding target function  $g_i$  is substituted for the internal variable  $y_i$  in every node function in which it appears. We stress that this restriction is an auxiliary construct for analysis, *not* an attempt to redesign the network under consideration.

Let  $C(\mathcal{N})$  denote the necessary and sufficient condition for combinationality, expressed as a function of the input variables.

The following theorem provides a means to compute this necessary and sufficient condition:

### Theorem 1

$$C(\mathcal{N}) = [f_1 \downarrow I_1] \cdot C(\mathcal{N}|_{f_1}) + \cdots + [f_n \downarrow I_n] \cdot C(\mathcal{N}|_{f_n}).$$

**Sketch of Proof:** For a network with cycles, we argue that, for each input assignment, at least one node function must evaluate to a definite boolean value independently of all the others. Indeed, if none of the functions evaluates to a definite boolean value, then no simplifications are possible and the network is not combinational. A function  $f_i$  evaluates to a definite boolean value independently of the others iff the marginal holds,

$$f_i \downarrow I_i.$$

Now, if a node function  $f_i$  evaluates to a definite boolean value, this value is given by the corresponding target function  $g_i$ . If we cut this node from the network, then the rest of the network must be combinational, that is,

$$C(\mathcal{N}|_{f_i})$$

must hold. Indeed, if a component of the network viewed in isolation is not combinational, then the entire network is not combinational.  $\square$

We illustrate the analysis with several examples. For Examples 1 and 2, consider the target functions,

$$\begin{aligned} d &= \bar{c}(\bar{a} + \bar{b}) + \bar{a}\bar{b} \\ e &= \bar{a}\bar{b}\bar{c} + \bar{b}(a + c) \\ f &= \bar{b}(\bar{a} + \bar{c}) + ab. \end{aligned}$$

### Example 1

Consider the network  $\mathcal{N}_1$ , shown in Figure 4. Note that

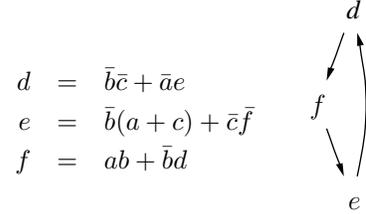


Fig. 4. Example: Network  $\mathcal{N}_1$ .

the dependency graph is a single cycle. The necessary and sufficient condition for combinationality is

$$\begin{aligned} C(\mathcal{N}_1) &= [d \downarrow e] \cdot C(\mathcal{N}_1|_d) + \\ & [e \downarrow f] \cdot C(\mathcal{N}_1|_e) + \\ & [f \downarrow d] \cdot C(\mathcal{N}_1|_f). \end{aligned}$$

The marginals are

$$\begin{aligned} d \downarrow e &= a + \bar{b}\bar{c} \\ e \downarrow f &= c + a\bar{b} \\ f \downarrow d &= b. \end{aligned}$$

Since we have a single cycle,

$$C(\mathcal{N}_1|_d) = C(\mathcal{N}_1|_e) = C(\mathcal{N}_1|_f) \equiv 1.$$

Thus,

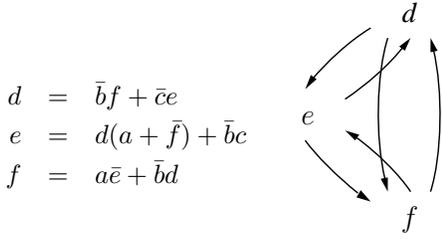
$$C(\mathcal{N}_1) = a + \bar{b}\bar{c} + c + a\bar{b} + b \equiv 1.$$

We conclude that the network is combinational for all input assignments.

### Example 2

Now consider the network  $\mathcal{N}_2$ , shown in Figure 5. Note that the dependency graph is the complete graph on three nodes. The necessary and sufficient condition for combinationality is

$$\begin{aligned} C(\mathcal{N}_2) &= [d \downarrow (e, f)] \cdot C(\mathcal{N}_2|_d) + \\ & [e \downarrow (d, f)] \cdot C(\mathcal{N}_2|_e) + \\ & [f \downarrow (d, e)] \cdot C(\mathcal{N}_2|_f). \end{aligned}$$



$$\begin{aligned} d &= \bar{b}f + \bar{c}e \\ e &= d(a + \bar{f}) + \bar{b}c \\ f &= a\bar{e} + \bar{b}d \end{aligned}$$

Fig. 5. Example: Network  $\mathcal{N}_2$ .

The marginals are

$$\begin{aligned} d \downarrow (e, f) &= bc \\ e \downarrow (d, f) &= \bar{b}c \\ f \downarrow (d, e) &= \bar{a}b. \end{aligned}$$

For the restriction  $\mathcal{N}_2|_d$ , we compute

$$\begin{aligned} e|_d &= \bar{b}(a + c) + \bar{a}\bar{c}\bar{f} \\ f|_d &= \bar{b}(\bar{a} + \bar{c}) + a\bar{e}. \end{aligned}$$

For this restriction, the marginals are

$$\begin{aligned} (e|_d) \downarrow f &= a + c \\ (f|_d) \downarrow e &= \bar{a} + \bar{b}\bar{c}. \end{aligned}$$

Now, recursively,

$$\begin{aligned} C(\mathcal{N}_2|_d) &= [(e|_d) \downarrow f] \cdot (1) + [(f|_d) \downarrow e] \cdot (1) \\ &= a + c + \bar{a} + \bar{b}\bar{c} \\ &= 1. \end{aligned}$$

Similarly, we compute

$$\begin{aligned} C(\mathcal{N}_2|_e) &= 0 \\ C(\mathcal{N}_2|_f) &= \bar{b} + c. \end{aligned}$$

Thus,

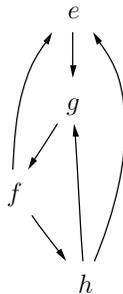
$$\begin{aligned} C(\mathcal{N}_2) &= (bc) \cdot (1) + (\bar{b}c) \cdot (0) + (\bar{a}b) \cdot (\bar{b} + c) \\ &= bc. \end{aligned}$$

We conclude that the network is combinational iff  $b = c = 1$ .

### Example 3

Consider the example in Figure 1,

$$\begin{aligned} e &= \bar{f}(a\bar{h} + c) + d\bar{h} + \bar{b} \\ f &= \bar{a}\bar{d}\bar{g} + a(\bar{b}d + bc) \\ g &= \bar{a}\bar{b}\bar{c} + \bar{h}(a\bar{e} + \bar{a}d + \bar{b}c) \\ h &= \bar{f}(a(c + d) + cd). \end{aligned}$$



The target functions for this network are

$$\begin{aligned} e &= a\bar{c}\bar{d} + d(c + \bar{a}) + \bar{b} \\ f &= ad(c + \bar{b}) + \bar{d}(\bar{a}\bar{b}\bar{c} + bc) \\ g &= \bar{a}\bar{c}(d + b) + c(\bar{a}\bar{b}d + \bar{d}(\bar{a}\bar{b} + ab)) \\ h &= \bar{a}\bar{b}\bar{c}\bar{d} + d(\bar{a}\bar{b}\bar{c} + \bar{a}c). \end{aligned}$$

The network is combinational iff the following condition holds

$$\begin{aligned} C(\mathcal{N}) &= [e \downarrow (f, h)] \cdot C(\mathcal{N}|_e) + [f \downarrow g] \cdot C(\mathcal{N}|_f) + \\ &\quad [g \downarrow (e, h)] \cdot C(\mathcal{N}|_g) + [h \downarrow f] \cdot C(\mathcal{N}|_h). \end{aligned}$$

Proceeding on a case basis:

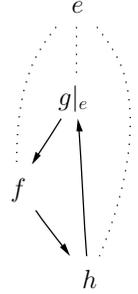
*Case I*

Suppose that  $e$  assumes a definite boolean value independently of  $f$  and  $h$ :

$$e \downarrow (f, h) = \bar{a}\bar{c}\bar{d} + \bar{b}.$$

Given this predicate, we obtain the sub-network  $\mathcal{N}|_e$  by substituting the target function for  $e$  into  $g$ :

$$\begin{aligned} f &= \bar{a}\bar{d}\bar{g} + a(\bar{b}d + bc) \\ g|_e &= \bar{a}\bar{b}\bar{c} + \bar{h}(c(\bar{a}\bar{d} + \bar{b}) + d(\bar{b}\bar{c} + \bar{a})) \\ h &= \bar{f}(a(c + d) + cd). \end{aligned}$$



Note that  $\mathcal{N}|_e$  contains a single cycle through nodes  $f, h$  and  $g|_e$ . We have three subcases:

1) suppose that  $f$  assumes a definite boolean value independently of  $g|_e$ :

$$f \downarrow (g|_e) = a + d,$$

2) suppose that  $g|_e$  assumes a definite boolean value independently of  $h$ :

$$(g|_e) \downarrow h = b(acd + \bar{a}\bar{d}) + \bar{c}(\bar{a}\bar{b} + \bar{a}b + \bar{d}),$$

3) suppose that  $h$  assumes a definite boolean value independently of  $f$ :

$$h \downarrow f = \bar{d}(\bar{a} + \bar{c}) + \bar{a}\bar{c}.$$

In each case, the assumption breaks the cycle. Assembling the three cases,

$$C(\mathcal{N}|_e) = [f \downarrow (g|_e)] + [(g|_e) \downarrow h] + [h \downarrow f] \equiv 1.$$

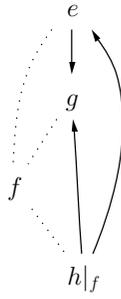
We conclude that the sub-network  $\mathcal{N}|_e$  is combinational.

### Case II

Suppose that  $f$  assumes a definite boolean value independently of  $g$ :

$$f \downarrow g = a + d.$$

Given this predicate, the sub-network  $\mathcal{N}|_f$  is obtained by substituting the target function for  $f$  into  $h$ ,

$$\begin{aligned} e &= \bar{f}(a\bar{h} + c) + d\bar{h} + \bar{b} \\ g &= \bar{a}b\bar{c} + \bar{h}(a\bar{e} + \bar{a}d + \bar{b}c) \\ h|_f &= \bar{a}\bar{b}c\bar{d} + d(\bar{a}b\bar{c} + \bar{a}c). \end{aligned}$$


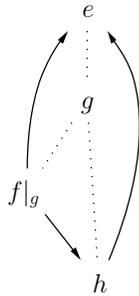
This sub-network is acyclic and hence combinational:  $C(\mathcal{N}|_f) \equiv 1$ .

### Case III

Suppose that  $g$  assumes a definite boolean value independently of  $e$  and  $h$ :

$$g \downarrow (e, h) = \bar{a}(\bar{d}(b + \bar{c}) + b\bar{c}).$$

Given this predicate, the sub-network  $\mathcal{N}|_g$  is obtained by substituting the target function for  $g$  into  $f$ ,

$$\begin{aligned} e &= \bar{f}(a\bar{h} + c) + d\bar{h} + \bar{b} \\ f|_g &= a(\bar{b}d + bc) + \bar{d}(\bar{a}b\bar{c} + bc) \\ h &= \bar{f}(a(c + d) + cd). \end{aligned}$$


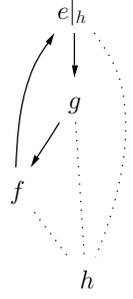
This sub-network is acyclic and hence combinational:  $C(\mathcal{N}|_g) \equiv 1$ .

### Case IV

Finally, suppose that  $h$  assumes a definite boolean value independently of  $f$ :

$$h \downarrow f = \bar{d}(\bar{a} + \bar{c}) + \bar{a}\bar{c}.$$

Given this predicate, the sub-network  $\mathcal{N}|_h$  is obtained by substituting the target function for  $h$  into  $e$ ,

$$\begin{aligned} e|_h &= d(\bar{a}\bar{c} + ac) + \bar{f}(a\bar{d} + c) + \bar{b} \\ f &= \bar{a}\bar{d}\bar{g} + a(\bar{b}d + bc) \\ g &= \bar{a}b\bar{c} + \bar{h}(a\bar{e} + \bar{a}d + \bar{b}c). \end{aligned}$$


Analyzing  $\mathcal{N}|_h$  in the same manner as in Case I, we find that  $C(\mathcal{N}|_h) \equiv 1$ .

Assembling the four cases,

$$\begin{aligned} C(\mathcal{N}) &= [e \downarrow (f, h)] \cdot (1) + [f \downarrow g] \cdot (1) + \\ & \quad [g \downarrow (e, h)] \cdot (1) + [h \downarrow f] \cdot (1) \\ &\equiv 1. \end{aligned}$$

Thus we conclude that the network in Figure 1 is combinational.

### B. Complexity

Malik has shown that the problem of analyzing a network to determine if it is combinational is co-NP-complete [8]. In the recursive decomposition of the necessary and sufficient condition for combinationality, one may encounter the same sub-network several times. Restriction is invariant to order so that for any  $i, j$ ,

$$(\mathcal{N}|_{f_i})|_{f_j} = (\mathcal{N}|_{f_j})|_{f_i}.$$

We need not recompute the condition for the same component encountered twice. For instance, in a network with nodes,  $f_1, f_2, \dots$ , we compute

$$C(\mathcal{N}) = (f_1 \downarrow y_1) \cdot C(\mathcal{N}|_{f_1}) + (f_2 \downarrow y_2) \cdot C(\mathcal{N}|_{f_2}) + \dots.$$

Recursively, we compute

$$C(\mathcal{N}|_{f_1}) = ((f_2|_{y_1})|_{y_2}) \cdot C((\mathcal{N}|_{f_1})|_{f_2}) + \dots,$$

and

$$C(\mathcal{N}|_{f_2}) = ((f_1|_{y_2})|_{y_1}) \cdot C((\mathcal{N}|_{f_2})|_{f_1}) + \dots.$$

We need not recompute  $(\mathcal{N}|_{f_2})|_{f_1}$ , as it is equal to  $(\mathcal{N}|_{f_1})|_{f_2}$ .

For a network corresponding to a complete graph on  $n$  nodes, the analysis requires on the order of  $n \cdot 2^n$  steps (there are  $2^n$  subsets of  $n$  nodes, each of which has  $n$  terms to evaluate). For less densely connected networks, the analysis is, of course, less complex.

### C. Timing

For timing analysis, Malik's approach is to transform a cyclic circuit into an equivalent acyclic one [8], much as Edwards proposes in recent work [7]. Timing information

is then obtained through functional timing analysis of the acyclic circuit.

We argue that timing analysis can be performed directly on a cyclic combinational circuit. In fact, timing analysis is closely related to combinationality analysis. In timing analysis, the goal is to find the longest sensitized path; in combinationality analysis, the goal is to ascertain whether there are any sensitized cycles (i.e., sensitized paths that bite their own tail).

In future work, we will address the issue of timing analysis in the context of synthesis as part of a complete methodology.

### III. SYNTHESIS ALGORITHMS

The goal in multilevel logic synthesis (also sometimes called random logic synthesis) is to obtain the best multilevel, structured representation of a network. The process typically consists of an iterative application of minimization, decomposition, and restructuring operations [3]. An important operation is **substitution**, in which node functions are expressed, or re-expressed, in terms of other node functions as well as of their original inputs. Our strategy is to introduce combinational cycles in the substitution phase. We have explored several approaches, including dynamic programming and branch-and-bound algorithms [10]. Here we discuss the interplay of analysis and synthesis in the design process.

The analysis method described in Section II is formulated recursively. Accordingly, it permits us to cache analysis results for common sub-networks through iterations of the search for a solution. Suppose that in the course of our search for a low-cost combinational solution we consider a network  $\mathcal{N}_1$  with node functions

$$f_1, \dots, f_n.$$

Analysis for combinationality entails evaluating the expression  $C(\mathcal{N}_1)$ , given in Theorem 1. Next, suppose that we consider a network  $\mathcal{N}_2$  with node functions

$$f'_1, \dots, f'_n.$$

Analysis entails evaluating  $C(\mathcal{N}_2)$ . Now suppose that some of the node functions in  $\mathcal{N}_1$  are identical to those in  $\mathcal{N}_2$ . Let  $\mathcal{S}$  be the subset of nodes that are identical:

$$\forall i \in \mathcal{S}, f_i \equiv f'_i.$$

The evaluation of  $C(\mathcal{S})$  figures in both  $C(\mathcal{N}_1)$  and  $C(\mathcal{N}_2)$ , and so it need not be repeated. If, in the process of evaluating  $C(\mathcal{N}_1)$ , we find that  $C(\mathcal{S}) = 0$ , then we rule out  $\mathcal{N}_1$  as well as  $\mathcal{N}_2$  (and all other networks that contain  $\mathcal{S}$ ). Otherwise, we find that  $C(\mathcal{S}) \equiv 1$ , and we need not re-evaluate it when evaluating  $\mathcal{N}_2$  (or any other network that contains  $\mathcal{S}$ ). We illustrate with examples.

#### Example 1

Consider again the example in Figure 1. Suppose that we have constructed the network for nodes  $f$  and  $g$  shown in Figure 6, assuming that nodes  $e$  and  $h$  are given. Analysis

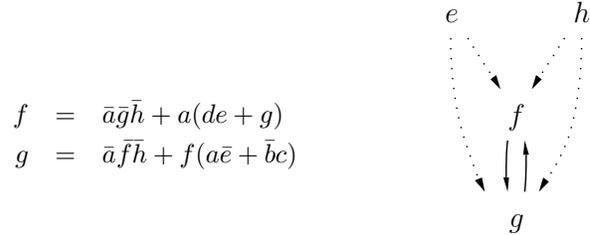


Fig. 6. A non-combinational component.

according to Theorem 1 tells us that this component is *not* combinational. Thus, we exclude this pair of node functions as candidates for  $f$  and  $g$ .

#### Example 2

Now suppose that we have constructed the candidates for nodes  $e, f$  and  $g$  shown in Figure 7.

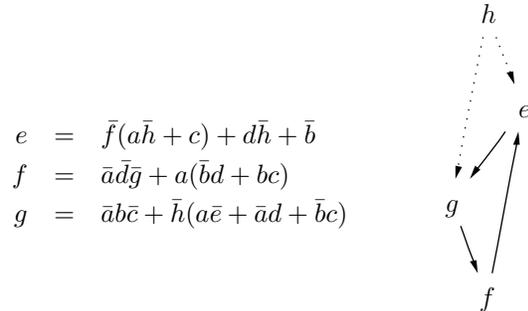


Fig. 7. A combinational component.

Analysis tells us that this component *is* combinational. We can proceed to select a node function for  $h$ . The candidates are

$$\begin{aligned} h_1 &= c(a\bar{d}\bar{e} + \bar{a}d) + d\bar{e} \\ h_2 &= \bar{f}(a(c+d) + cd) \\ h_3 &= \bar{g}(d(b\bar{c} + \bar{a}) + \bar{b}c) \\ h_4 &= c\bar{f}(a+d) + d\bar{e} \\ h_5 &= \bar{f}\bar{g}(c+d) \\ h_6 &= c\bar{f}\bar{g} + d\bar{e}. \end{aligned}$$

When analyzing networks constructed with these candidates for  $h$ , we need not re-evaluate the component  $e, f, g$  from Figure 7. We find that if  $h_2$  is combined with this component, it yields a combinational network (that shown in Figure 1).

### IV. RESULTS

In [10], we present synthesis results for benchmark circuits [1], [2]. We note that solutions for many of the examples contain dozens or even hundreds of cycles. The dependency graph of the cyclic solution for one of the Espresso benchmark circuits, **exp**, is shown in Figure 8.

