1-1-1984

Consistency Testing for Data-Flow Circuits

Chu S. Jhon
Seoul National University

Robert M. Keller
Harvey Mudd College

DEADLOCK ANALYSIS IN THE DESIGN OF DATA-FLOW CIRCUITS*

Chu S. Jhon

Electrical and Computer Engineering Department
University of Iowa, Iowa City, Iowa

Robert M. Keller

Computer Science Department
University of Utah, Salt Lake City, Utah

ABSTRACT

One means of making VLSI design tractable is to proceed from a high-level specification of a circuit in terms of functionality, to the circuit level. A notable error which may occur in a top-down design starting with a data-flow graph representation of a circuit is a design inconsistency due to deadlock. This paper attempts to further develop the theoretical basis for algorithms which analyze the deadlock property of circuits on the basis of their data-flow graph representations. A systematic scheme to verify the absence of deadlock in data-flow graphs is also presented.

1. INTRODUCTION

Data-flow graphs provide a modular approach to the design of VLSI systems from their high-level specifications[2]. In a data-flow graph, a node typically represents a function from streams (i.e. infinite sequences) of input values to streams of output values. That is, the node repeatedly absorbs input values and produces output values as specified by the function. An arc of the data-flow graph denotes the flow of the output value of one function as an input to another. Conceptually, an arc may retain arbitrarily many values when there is an "imbalanced consumption" of replicated streams by nodes.

A data-flow graph incurring deadlock is a design fault wherein an intended communication between a pair of nodes never completes, due to malformed interconnection. A complete analysis of deadlock in a data-flow graph requires that the input/output behavior of every node be known. However, the general problem for an arbitrary set of functions is doomed to be intractable. Hence, we restrict ourselves to a limited set of function schemes to specify nodes shown below:

1. An APPLY-TO-ALL version of a function "f" (denoted by f\\) which repeatedly absorbs one value each from all the inputs and then produces the output value by applying f to this set of absorbed values. For example, +\\[123...,123...]=246....
2. The COND function, which has three inputs, and repeatedly absorbs the value arriving from the first input. It then passes the value arriving from the second (third) input to the output, depending upon whether the value absorbed from the first input is "1" ("0"). For example, COND[1010...,13...,24...]=1234....
3. The T-GATE function, which has two inputs, and passes the value arriving from the second input to the output only when the value absorbed from the first input is "1"; otherwise, the values arriving from both the inputs are merely absorbed. For example, T-GATE[0101...,1234...]=24....
4. The fanout point, which has one input and two outputs, and replicates a stream.

2. DEADLOCK IN DATA-FLOW CIRCUITS

By a data-flow circuit, we mean a circuit which can be synthesized from a given data-flow graph by replacing every node with a functionally-equivalent circuit module, and every arc with a set of wires (possibly together with serially connected buffer slots). Deadlock in a data-flow circuit is informally defined as a state in which the computation of some circuit module cannot be completed, yet must yield any further output "token" (hereafter, we shall refer to a valid value as a token). We classify deadlock into two main categories: buffering-independent deadlock (BID), which is caused by the inability of a module to produce any more tokens because its necessary input tokens never become available; and buffering-dependent deadlock (BDD), which is caused by the inability of a module to produce any more tokens because the module receiving them is unable to absorb further tokens.

An example of BID is shown in Fig.1(a), and is self-explanatory. An example of BDD is shown in the data-flow graph of Fig.1(b). Here the placement of only finitely many buffer slots on a circuit module for arc "a" may make it impossible to synthesize a data-flow circuit from this graph. The computation of the ith token forces the module to retain i-1 input tokens to preserve the same functionality. Hence, the module must have arbitrarily many buffer slots in cases where the data-flow circuit needs to produce arbitrarily many output tokens. Were this arbitrary buffering capacity possible, there would be no deadlock. But in a physical circuit, the buffer size is limited.

3. ANALYSIS OF DATA-FLOW CIRCUITS

This section presents a conceptual model which simplifies deadlock analysis in data-flow graphs. Constructive schemes to derive such a model while verifying the absence of deadlock, are described.

In the asynchronous computation of our data-flow graphs, the computation of nodes may be enabled simultaneously in any arbitrary order. In a state transition sense, such an asynchronous computation is deterministic, commutative, and persistent. Keller[1] proves that such a computation satisfies the Church-Rosser property, which indicates that

* This work was supported in part by the Semiconductor Research Corporation (contract no. 83-01-003) and the National Science Foundation (MCS-8106177).

for any two states reachable from a common state, there is a common state reachable from both. Therefore, if a specific asynchronous computation (including a synchronous computation which is a special case) reaches a deadlock state, then any other asynchronous computation may also. Thus, it suffices to deal only with the class of synchronous computations.

We shall explain deadlock behavior of data-flow circuits through the notion of "synchronous token-flow" that can be constructively identified from their data-flow graph representations.

Definition 1 : A synchronous token-flow for a data-flow graph is the flow of tokens resulting from the successive firing (application), during each clock period, of the largest set of nodes satisfying the following conditions:

a) A node is fired iff (1) all the necessary input tokens are available and (2) the output token produced, if any, can be absorbed by an output destination or by a reachable node, which when fired, merely absorbs the input tokens (e.g. T-GATE). Here by an available token, we mean that the token is present sometime during the clock period, not necessarily only at the beginning.

b) At each node, all available input tokens which are not consumed are retained in their input arcs so as to be available in the next clock period.

c) No node can be fired more than once during a clock period.

Note that the largest set of nodes may vary from clock period to clock period.

The synchronous token-flow behavior of each arc in a data-flow graph can be represented by a "pattern stream" defined as follows:

Definition 2: A pattern stream X is a stream of bits which is associated with each arc of a data-flow graph such that in the synchronous token-flow for the graph, if the arc passes a token during the ith clock period, then the ith element X_i is 1; otherwise, it is 0.

Fig.4(c) illustrates a data-flow graph which computes the Fibonacci stream 112358..., along with the associated pattern stream for each arc. In the synchronous token-flow for the graph, during the first clock period only the COND node at the top is fired, during the second period all the COND nodes are fired, and from the third onward all the nodes are fired. For brevity, we shall denote a stream as a regular expression whenever possible. For example, $0(1)^*$ denotes 0111....

Owing to the modular nature of the synchronous token-flow defined above, pattern streams can be constructively derived. For example, the pattern streams of the data-flow graph which is derived by merging two graphs A and B as shown in Fig.2 are systematically derivable. We emphasize that the synchronous token-flow for the merged graph may warrant more clock periods than that of graph A or B. The additional periods are needed whenever graph A cannot consume a token arriving from graph B during a particular clock period, or graph B is incapable of producing a token that is required by A. This scheme is made more clear in the description of Function F which derives pattern streams for the merged graph of Fig.2(b).

Function F:

INPUT: X; pattern stream for an arc in graph A (B). Pattern streams P and Q of Fig.2(a).

OUTPUT: V; pattern stream for the corresponding arc in the merged graph of Fig.2(b).

PROCESS:

1. i:=1,j:=1,k:=1
2. if $P_i=Q_j$ then $V_k:=X_i(X_j)$,i:=i+1,j:=j+1,k:=k+1
   if $P_i>Q_j$ then $V_k:=0(X_j)$,j:=j+1,k:=k+1
   if $P_i<Q_j$ then $V_k:=X_i(0)$,i:=i+1,k:=k+1
3. Repeat STEP 2.

Given the boolean streams for the relevant inputs (e.g. first input of COND node), the pattern streams for our primitive nodes are determined as shown in Fig.3, without any constraints on their input sources and output destinations. These associated pattern streams provide a constructive basis to derive pattern streams for a general data-flow graph. As an example, pattern streams for the graph of Fig.4(a) can be constructively derived using these streams along with Function F.

Similarly, pattern streams for a data-flow graph constructed by adding a fanout point to a general graph, as shown in Fig.5, are derived by Function G defined below:

Function G:

INPUT: P and Q of Fig.5(a).

OUTPUT: R of Fig.5(b) (others are unchanged).

PROCESS:

1. i:=2, if $P_1=Q_1=0$ then $R_1:=0$ else $R_1:=1$
2. if $\max(\sum_{j=1}^{i} P_j, \sum_{j=1}^{i} Q_j) > \sum_{j=1}^{i-1} R_j$ then $R_i:=1$ else $R_i:=0$
3. i:=i+1, go to STEP 2.

The acyclic connection shown in Fig.2 may introduce BID. If graph B is always capable of producing tokens required by graph A, the merged graph is free from BID. This condition is represented as $C(P) \leq C(Q)$, wherein C(X) denotes the number of 1s occurring in the stream X, and can be infinite.

Another case of BID may be introduced in the construction of a cyclic data-flow graph, as shown in Fig.6. BID may arise in cases where the arc associated with Q is free of tokens, and the arc associated with P can carry a token only after that associated with Q carries a token. We introduce a notion of "accumulatively less than or equal to" to specify a condition for the absence of BID.

Definition 3: For two infinite streams P and Q, P is said to be accumulatively less than or equal to Q iff the sum of first i elements of P is less than or equal to the corresponding sum of Q for all i.

Using the above explanation, we arrive at:

Theorem 1: The cyclic graph of Fig.6(b) constructed by introducing a cyclic connection to the deadlock-free graph of Fig.6(a) is free from BID, if the first element of the pattern stream Q is 0, and the stream derived by removing the first element of Q is accumulatively less than or equal to the pattern stream P.

Wadge's "check sum test"[3] follows a special case of Theorem 1.

For example, graphs of Fig.4(b) and (c) are free from BID, since first elements of both $0(1)^*$ and $00(1)^*$ are 0s, and $0(1)^*$ is accumulatively less than or equal to both $0(1)^*$ and $(1)^*$. We note that in this case, a pattern stream associated with each arc is not changed due to the cyclic connection.

In the addition of a fanout point to a deadlock-free graph as shown in Fig.5 and 6, BDD may be introduced. BDD occurs due to a limit on the buffering capacity of some outputs for the added fanout point. To express a condition for the absence of BDD, we define the "accumulative boundedness":

Definition 4: Two infinite streams P and Q are said to be accumulatively bounded iff there exists a positive integer N such that the absolute value of the difference between the sum of the first i elements of P and that of Q is less than or equal to N for every non-negative integer i.

N denotes the maximal number of tokens which can be retained in output arcs of added fanout points in the synchronous token-flow with pattern streams P and Q of Fig.5 and 6.

Theorem 2: Data-flow graph shown in Fig.5(b) (Fig.6(b)) is free from BDD, if pattern streams P and Q are accumulatively bounded and the graph of Fig.5(a) (Fig.6(b)) is deadlock-free.

As an example, the graph of Fig.4(c) is found to be free from BID by applying Theorem 2.

4. A SYSTEMATIC DEADLOCK ANALYSIS SCHEME

Deadlock properties of a data-flow circuit are determined by the characteristics of its input sources and output destinations. The scheme proceeds with the construction of a data-flow graph, which consists of arcs that are inputs to all output destinations of a given graph to be tested, along with the associated pattern streams. These pattern streams are specified so as to denote tokens to be consumed by the corresponding output destinations depenendently of each other. We then construct a new graph with associated pattern streams by applying the constructive rules of section 3, to connect a primitive stream-based function, a fanout point, and an arc (which is output from an input source) to the old graph, while checking for the introduction of deadlock in the new graph. This process is repeated until no further rule is applicable, in which case failure of testing is reported, or the graph being tested is constructed, in which case there is no deadlock.

The above scheme does not terminate when pattern streams for a given graph are not effectively presented. One type of pattern stream which is effectively derivable is that represented by "deterministic regular expression", defined as a regular expression consisting of a finite sequence followed by the *-closure of a finite sequence (hence there is no union operator). If a data-flow graph has deterministic regular expressions for both boolean streams of data-dependent decision nodes, and pattern streams for arcs to/from output destinations/input sources, then the scheme terminates.

5. CONCLUSION

The contribution of this paper is to help make the data-flow approach a more realistic alternative to contemporary VLSI design methodologies by providing a basis for systematically ensuring the absence of deadlock in circuits at the data-flow specification level. One avenue for future research is to devise a means of extending the limited framework of deterministic regular expressions so that the deadlock property of less-constrained data-flow specifications can be algorithmically detected. Another avenue is to extend our schemes to include a wider class of data-flow graphs.

6. REFERENCES

[1] R.M. Keller, "A fundamental theorem of asynchronous parallel computation," In Lecture Notes in Computer Science, Vol. 24, Springer-Verlag, pp. 102-112, 1975.

[2] R.M. Keller, G. Lindstrom, and S. Patil, "Data-flow concepts for hardware design," In IEEE COMPCON 80, pp. 105-111, Feb. 1980.

[3] W.W. Wadge, "An extensional treatment of data-flow deadlock," In Lecture Notes in Computer Science, Vol. 70: Semantics of Concurrent Computation, G. Kahn, ed., Springer-Verlag, pp. 285-299, 1979.

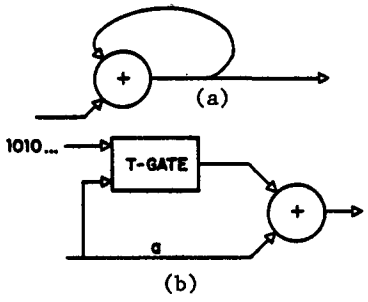

Fig.1

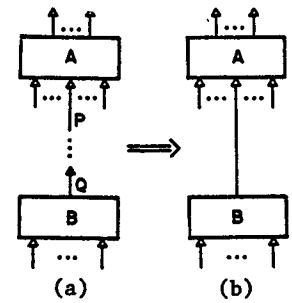

Fig.2

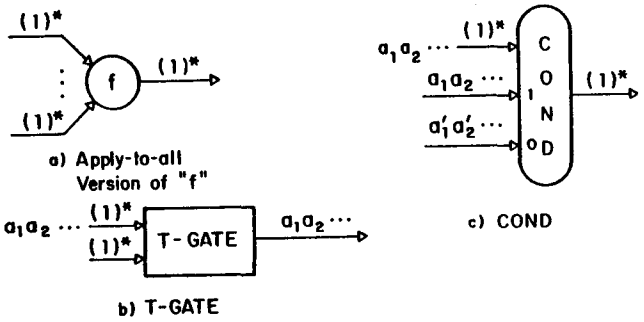

Fig.3

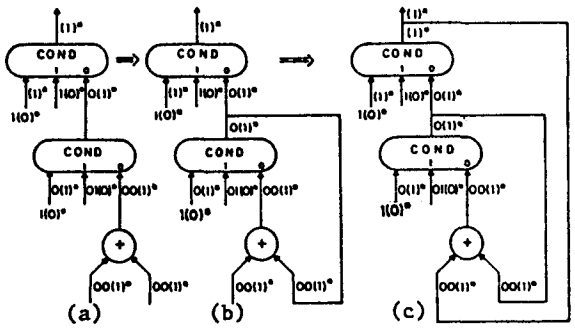

Fig.4

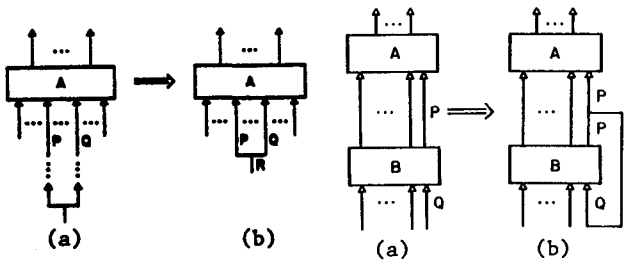

Fig.5

Fig.6