Diagnosing Circuits with State: An Inherently Underconstrained Problem

Walter Hamscher
Randall Davis

Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Square
Cambridge, Mass 02139

"Hard problems" can be hard because they are computationally intractable, or because they are underconstrained. Here we describe candidate generation for digital devices with state, a fault localization problem that is intractable when the devices are described at low levels of abstraction, and is underconstrained when described at higher levels of abstraction. Previous work [1] has shown that a fault in a combinatorial digital circuit can be localized using a constraint-based representation of structure and behavior. In this paper we (1) extend this representation to model a circuit with state by choosing a time granularity and vocabulary of signals appropriate to that circuit; (2) demonstrate that the same candidate generation procedure that works for combinatorial circuits becomes indiscriminate when applied to a state circuit modeled in that extended representation; (3) show how the common technique of single-stepping can be viewed as a divide-and-conquer approach to overcoming that lack of constraint; and (4) illustrate how using structural detail can help to make the candidate generator discriminating once again, but only at great cost.

Introduction

Faults in combinatorial digital circuits can be localized using a constraint-based representation of structure and behavior. This fault localization procedure, *candidate generation*, is reviewed below. The procedure is general and should apply to circuits with state: we have extended the constraint-based representation to include these devices. A key feature of the extended representation is the use of layers of temporal granularities. In this paper we show a simple example of such a multilayered description.

But, having extended the representation, we show that the same diagnostic procedure that works well for combinatorial circuits becomes indiscriminate when applied to state circuits. Intuition tells us that circuits with state are more difficult to diagnose than combinatorial ones: we show that this intuition is correct by presenting a computational view of the candidate generation process. Intuition also tells us that single-stepping a circuit is a good way to localize faults: this intuition too turns out to have firm computational grounds. Finally, we show that knowledge about the substructure of a device can provide considerable additional discriminatory power.

Candidate Generation

Given a device exhibiting faulty behavior, we wish to determine which of its subcomponents could be responsible for the misbehavior. We call these components *candidates*. The most effective diagnoses are those which propose the fewest alternative candidates, in which the candidates represent the least complex hardware, and in which the candidates' hypothesized misbehaviors are most specific.

We represent each component of a circuit as a *module* [3]. Modules have substructure, composed of modules connected by wires. The primitive modules of the system are logic gates. The behavior of each module can be expressed as a constraint [4, 5] on the values at its terminals. The constraint is itself composed of a set of rules spanning the device. For example, the behavior of a two-input NAND-gate can be described as a constraint composed of the following rules:

If both inputs are 1, the output must be 0.
If one input is 0, the output must be 1.
If the output is 0, both inputs must be 1.
If the output is 1 & one input is 1, other input is 0.

For the sake of simplicity in this discussion, we assume that faults occur only in modules. This allows us to ignore some uninteresting details without affecting the essential nature of candidate generation. The process is best understood by considering a simple example. Figure 1 shows a combinatorial circuit that computes F = AC + BD and G = BD + CE, with inputs 3, 3, 1, 1, and 3. The modules' behavioral constraints tell us that if all the modules are working, we can expect the outputs at F and G to be 6 and 6. Imagine, however, that we observe the outputs in the actual device to be 5 and 6. Which components could be responsible for this discrepancy?

We find the potential candidates by tracing backward from the discrepant output F. All modules that contributed to that output are potential candidates: in this case, MULT-1, MULT-2, and ADD-1. To find out which of those potential candidates can account for all the behavior observed, we consider each one in turn, suspending its constraint [2], and asking whether the resulting network is now consistent with the inputs and observations.[•]

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's AI research on hardware troubleshooting is provided in part by a research grant supplied to MIT by the Digital Equipment Corporation, and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505.

[•] The same procedure works under weaker assumptions, e.g. assuming two points of failure we suspend pairs of constraints.

In this example we find that MULT-2 is not a consistent candidate. Suspending its constraint leads to a contradiction: the inputs at A-E and the observations at F and G are inconsistent with correct behavior of the remaining four modules. MULT-1 is, however, a consistent candidate: it could have misbehaved by having 3 and 1 as inputs and 2 as its output. ADD-1 is also a consistent candidate: it could have had a 3 and a 3 as inputs and a 5 as its output.

Having deduced these possible misbehaviors for ADD-1 and MULT-1, we can in effect construct a new behavior for each candidate. This information is important: it says not only *whether* the module could be failing, but if it is, *how* it's failing. Each new test then supplies additional information about the module's (mis)behavior, in effect building up a truth table showing how the module must be (mis)behaving.

Representation of Circuits with State

We have seen that the behavior of purely combinatorial devices can be modeled in a natural way using constraints. We model devices with state by extending our representation to include constraints that span time. To do this, we also need to extend our description of signals from single numbers to sequences of (*value,time*) pairs, with each pair denoting the signal's value at an instant. The behavior of a flipflop is then described as:

If the clock input is 0 at time *t-1* and 1 at time *t*
  Then output Q at time *t + 1* equals D at time *t*
  Else output Q at time *t + 1* equals Q at time *t*.

Since a hierarchic representation of time is as useful as a hierarchic representation of structure, we describe the behavior of modules using several different granularities of time. The most basic unit of time is the switching time of a gate. Other, coarser, units are less obvious and one of the difficulties we encounter lies in choosing the appropriate levels of temporal abstraction. One of the secondary contributions of this paper is its attempt to define a number of levels that appear to be both useful and intuitive in the current domain.

With a hierarchic representation of time, the behavior of a module can be described at several different levels of abstraction. For example, at the finest level of detail, a NOR gate can be modelled as having a unit delay. This would be a appropriate in an asynchronous feedback circuit, since the delay is important in understanding the behavior of the circuit.

But a NOR gate in the combinatorial part of a properly designed clocked circuit can be modelled as having *no* delay; indeed, a "properly designed" clocked circuit is one in which the clock period is longer than the maximum delay to quiescence of any combinatorial component.

Similarly, it is appropriate in some contexts to model a JK flip-flop as imposing a unit delay between its data inputs and its outputs; it may also be necessary to model its behavior at the gate-delay level, in which case the delay between the J and K inputs and the outputs may be 4 or more units. We maintain alternative descriptions for the same type of device, as well as explicit mappings between those descriptions at different granularities.

Different granularities of time also lead us to make use of symbolic values for signals in cases where transitions, rather than quiescent values, are important. For example, the clock input of a rising-edge-triggered flipflop is described using the values 0 and +P, where +P denotes a rising edge followed by a falling edge. This abstraction allows us to describe the flipflop's behavior in part as:

If the clock input is +P at time *t*,
  Then output Q at time *t + 1* is same as D at time *t*.

A more complex example is shown in Figure 2, which shows a two-bit register "TBR" that clears itself whenever 1's are clocked into both its flipflops. Figure 3 shows, at three different time granularities, TBR's response to a series of changes on its inputs. We describe the behavior of this device at these multiple levels, to show how behavior at fine temporal granularities maps onto behavior at coarser granularities.

Figure 3a shows the changes on TBR's signals at the lowest time granularity, using delay of a gate as the basic unit (the small tick marks on the time axis). To represent behavior at this level, we use the flipflop behavior description given above and a description for the AND gate that imposes a unit delay between the inputs and outputs:

If both inputs are 1 at time *t*, output is 1 at time *t + 1*.
If an input is 0 at *t*, the output must be 0 at *t + 1*.
If the output is 1 at *t*, both inputs must be 1 at *t-1*.
If the output is 0 at *t* and an input is 1 at *t-1*,
  the other input is 0 at *t-1*.

At this lowest level, every transition on signals in TBR is visible; this is a fairly continuous view of the device's behavior.

Figure 3b shows the behavior at the next coarsest level, showing the values of all signals only at the instants of possible clock transitions (the large tick marks on the time axis). The values of D0, D1, and Q1 have been sampled at those points, producing the values at the next level. Here the basic unit is the cycle time of the clock and the internal CLR signal has become invisible. This yields the external behavior of the device, described as:

If CLK is +P at time *t*,
  Then Q1 at time *t + 1* is AND(D1 at *t*,NOT(D0 at *t*))
  Else Q1 at time *t + 1* is Q1 at time *t*.

There are two key features to this mapping between levels. First, the *longest delay until quiescence* at the finer level determines how many fine-grain units correspond to one coarse-grain unit. In this example, 5 units at the finer level maps onto a unit delay at the coarser level, because the behavior of the device normally requires at most 5 of the fine units to reach quiescence.

Second, events whose duration is shorter than the current level of granularity are not represented. In this example the duration of the momentary "1" value of Q1 after the second clock transition is not represented at the coarser level, because it falls entirely within one unit of time at that level, and hence is never "seen".

Figure 3c shows the behavior at the coarsest level of granularity, representing TBR's external signals only at those instants when the clock makes a positive-going transition.

This hides the details of the CLK signal: we can see the changes that occur when the clock makes its transition, but have no idea of how long it is between transitions. We represent this behavior as:

Q1 at time *t + 1* is the AND of D1 at *t* and NOT(D0 at *t*)).

Candidate Generation Applied to A Circuit With State

We can combine the candidate generation technique reviewed above with this representation of circuits with state as a first step in diagnosing those circuits.

Consider for example a 4-bit sequential multiplier MULT shown in Figure 4. MULT has two input registers A and B, and an 8-bit accumulator register Q. When the INIT signal is high, the A and B inputs are loaded into the A and B registers and the Q register is cleared. On each clock pulse, the A register shifts down, the B register shifts up, and the Q register accumulates the product. After four clock cycles the Q register contains the product A*B.

If we load the inputs 6 and 9 into the A and B registers on the first clock cycle, we expect to see 54 in register Q four clock cycles later. Suppose, however, we observe 58. We want to find out which components could have failed in such a way as to produce this symptom.

To illustrate how constraint suspension can be used to find the consistent candidates in the circuit, we use a standard technique of replicating the multiplier over five clock cycles (as in Figure 5), producing a snapshot of the circuit behavior at each cycle. The ovals in the diagram represent signal values. Each signal is replicated five times in the diagram; each of these ovals represents the value of the signal at each of five clock pulses. The snapshots are linked by connections that suggest the transmission of register values from one time period to the next. The diagram shows that we expect the successive values of Q to be 0, 0, 18, 54, and 54. The CLK signal is implicit, just as in the layered temporal representation described above. The INIT signal is not shown for simplicity's sake; it makes no contribution to the following analysis.

Suppose we observe only the final contents of Q, which is 58 instead of the expected 54. Tracing back from the expected value of Q, we find that all five components of MULT were supposed to contribute to that output. Thus all five components are potential candidates. To check whether these modules are consistent candidates under the single point of failure assumption, we suspend each of their behaviors in turn, by removing the constraint that each component imposes in all time slices, and check to see whether its removal is consistent with the incorrect output. Doing this, we find that register A is not a consistent candidate, since there is no sequence of 1's and 0's that we can assign to the least significant bit of register A that could explain the result of 58.

But all four of the other devices are consistent candidates. Worse, we are not able to deduce any specific misbehavior for them. The reason for this can be seen by looking at the constraint graph in Figure 5. If we know only the inputs at the top and the single output at the bottom, then suspending the constraint for the driver or the adder disconnects the graph and the inputs become irrelevant. Suspending the B register constraint leaves an almost-disconnected graph. In this case the values of the B register at *t2* and *t3* can be any pair that sums to 58.

The candidate generator has become indiscriminate: four of five modules in the multiplier are candidates. This is significant, because this example is not a pathological one. The problem is intrinsic to devices with state: hypothesizing a failure in a part means *removing constraints in many time-slices*. This in turn tends to leave large gaps in which it is impossible to deduce what actually happened.

Intuition tells us that circuits with state are hard to diagnose; this intuition now has firm computational ground: circuits with state are hard to diagnose in part because the problem is often inherently underconstrained.

Introducing More Visibility

Intuition also tells us that single-stepping state circuits and observing the intermediate results vastly reduces the problem. We can now see why.

Suppose we are able to observe the contents of Q at each of the five clock cycles, and we observe that it contains 0, 1, 20, 57, and 58 instead of 0, 0, 18, 54, and 54 as expected. This provides two important sources of power. First, we have in each slice a strictly combinatorial device. Since the subproblem of generating candidates in combinatorial circuits is typically sufficiently constrained, we expect to generate a more restricted set of candidates in each slice. Second, we have four I/O pairs, in effect four tests of the device. Since we are assuming a single point of failure, to be a candidate a component must be consistent with the observations in all four slices. This too will help to restrict the number of candidates generated.

If we examine Figure 4, we find discrepancies at Q in the first through fourth time slices. In each slice we trace backwards from Q, yielding four sets of potential candidates. We intersect these sets to find the candidates consistent with the information in all four slices: the Q register, A register, adder, and driver. The B register was eliminated from consideration because its misbehavior could not explain the discrepant output of Q at t_2. Having determined the potential candidates by tracing back from discrepancies and enforcing consistency across time slices, we now determine which of these modules is consistent with the observations.

- As before, register A is not a consistent candidate. (There is no set of assignments to its least significant bit over four time slices that yields the observed contents of Q when the B input is 9 and all other constraints are operating.)
- The driver is a consistent candidate, its misbehavior can be partially described by the following truth table:

CTL	IN	OUT	
0	9	1	(should be 0)
1	18	19	(should be 18)
1	36	37	(should be 36)
0	72	1	(should be 0)

Table 1: Truth Table of Misbehaving Driver

- The adder is a consistent candidate. (Note that removing its constraint in all four time slices completely disconnects each of the observed values from the inputs and from each other; for this reason a faulty adder would explain *any* observations.) It has the following misbehavior:

INPUT-1	INPUT-2	OUTPUT	
0	0	1	(should be 0)
18	1	20	(should be 19)
36	20	57	(should be 56)
0	57	58	(should be 57)

Table 2: Truth Table of Misbehaving Adder

- The Q register is a consistent candidate. As with the adder above, removing its constraint disconnects our observations from the inputs, so that this device's failure could explain anything. Its truth table is:

INPUT at *t*	OUTPUT at *t + 1*	
0	1	(should be 0)
19	20	(should be 19)
56	57	(should be 56)
57	58	(should be 57)

Table 3: Truth Table of Misbehaving Q Register

We have gained important information in the form of truth tables that describe how each candidate could have failed so as to produce the observed symptoms. Still, even with complete visibility of the outputs, under a strong set of assumptions, we are unable to distinguish among 3 of the 5 components of this device. We need yet more information.

Hierarchic Diagnosis

The only remaining source of information is the substructure of the candidates. We can use this information by applying the candidate generation procedure to each of the remaining consistent candidates. Note that we take this step with some reluctance: from a practical point of view, using structural information is expensive because the number of potential and consistent candidates tends to increase dramatically, even though the complexity of the individual candidates decreases.

- The Q register is built from eight D-flipflops sharing their clock and clear inputs (Figure 6). We use the behavior deduced for Q and map down from our coarse-grained temporal view to the next level of temporal detail, at which the clock signal is visible. Applying candidate generation to Q, we find that there is no single flipflop whose failure could explain the observed misbehavior of Q. That is, there is no single flipflop whose failure could produce the symptoms shown in Table 3.(The discrepancies in Q occurred in the three low-order bits. Each of these discrepancies results in a set of potential candidates. But the intersection of these sets is null.)

- The driver is composed of eight AND gates sharing a control input (Figure 7). Proceeding as above, we find that one AND gate --- the one enabling the least significant bit --- is a consistent candidate: if this AND gate's output is always 1 no matter what its inputs, we get the observations of Table 1.

- The adder is composed of eight single-bit adder slices. (Figure 8). The least significant slice is a consistent candidate: its output, viewed as a 2-bit integer, is always 1 greater than it should be.

The candidate set has now been reduced to only two modules: one AND gate in the driver and one bit-slice in the adder. Given the symptoms available, and excluding the possibility of internal probes, it is not possible to distinguish between the two.

This result illustrates the power of information about substructure in refining candidate generation, both in the number of candidates and in their complexity. The single point of failure assumption and single-stepping of state circuits, while reducing the possible candidates considerably, were still not sufficient to reach a satisfactory diagnosis.

As a final note of the power of this approach, note that the final candidate set was reached *even without assuming that the fault was nonintermittent.* The nonintermittency assumption says that the faulty module is failing consistently, i.e., given the same inputs, it produces the same (incorrect) output. In our terms this amounts to insisting that the behaviors deduced for a candidate be consistent across all the time slices, i.e., the tables like those shown in the previous section have to be self-consistent.

We were able to rule out many of the potential candidates using a weaker form of consistency implied by the single point of failure assumption: we required only that *some* behavior of each candidate be able to account for the discrepancies in all the time slices. It might, for example, have been the case that the adder could be a candidate only if it added 0 and 0 to get 1 in time slice 1, and added 0 and 0 to get 0 in time slice 4. Even with this weaker form of consistency, we were able to constrain the candidates we generated simply because they could not account for the discrepancies in all four time slices under any behavior, intermittent or not.

Limitations and Future Work

The temporal abstraction described and used here is limited: short events are invisible at higher levels of abstraction, yet often hardware failures involve short events. Consider for example a gate which has failed by slowing down, rather than failing altogether. This will cause incorrect results only when this slowness causes some signal to be sampled too soon, before it has a chance to change to its correct value. If this misbehavior is observed at a coarse temporal granularity, it may appear to be intermittent. Any level or kind of abstraction, in fact, falls prey to faults that it can represent, but not derive: a coarse-grained model of time can represent hazards and races as intermittent faults, but it can't distinguish between devices that have slowed down and ones that are genuinely unpredictable. This fact puts a premium on careful definition of the mappings between layers of temporal granularities. One goal of this research is to further investigate the nature of hierarchic diagnosis using these temporal hierarchies in addition to structural ones.

Conclusion

Combinatorial circuits can be modeled in a natural way using constraints and this representation can be used for generating candidate components. Circuits with state can also be modeled by constraints if the representation is extended to use multiple levels of time granularity. Intuition tells us that circuits with state are more difficult to diagnose than combinatorial ones, and we have shown a computational reason for this: *when less than complete state visibility is available, candidate generation is inherently underconstrained and therefore indiscriminate*. Intuition also tells us that single-stepping a suspect state circuit is a good way to localize faults: we showed that this intuition too turns out to have firm computational grounds: single stepping allows us to view the problem as a more constrained problem, that of diagnosing a combinatorial circuit. Finally, by using information about devices' internal structure and viewing devices at a fine temporal granularity, specific diagnoses can be obtained even for devices with state.

Acknowledgments

Howard Shrobe, Ramesh Patil, Thomas Knight, and all the members of the MIT AI Lab's Hardware Troubleshooter group contributed to the content and presentation of this research.

References

[1] R. Davis.
Diagnosis Via Causal Reasoning: Paths of Interaction and the Locality Principle.
In *Proceedings of AAAI-83*, pages 88-94. AAAI, August, 1983.

[2] R. Davis.
Reasoning from Structure and Behavior.
1984.
To appear in *Artificial Intelligence*.

[3] R. Davis and H. Shrobe.
Representing the Structure and Behavior of Hardware.
IEEE Computer 16(10):75-82, October, 1983.

[4] DeKleer, J., and G. J. Sussman.
Propagation of Constraints Applied to Circuit Synthesis.
International Journal of Circuit Theory 8(2):127-144, April, 1980.

[5] G. L. Steele.
The Definition and Implementation of a Computer Programming Language Based on Constraints.
Technical Report AI-TR-595, MIT, 1980.

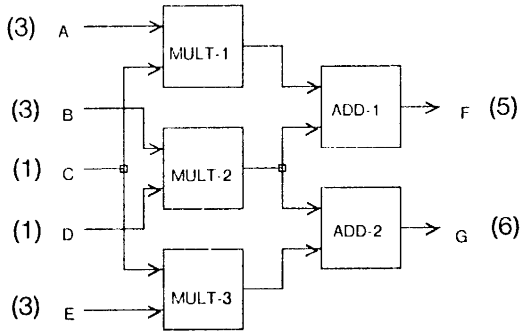

Figure 1: Combinatorial Circuit Example

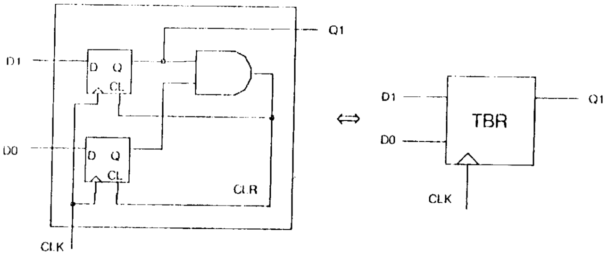

Figure 2: Self-clearing Two-bit Register

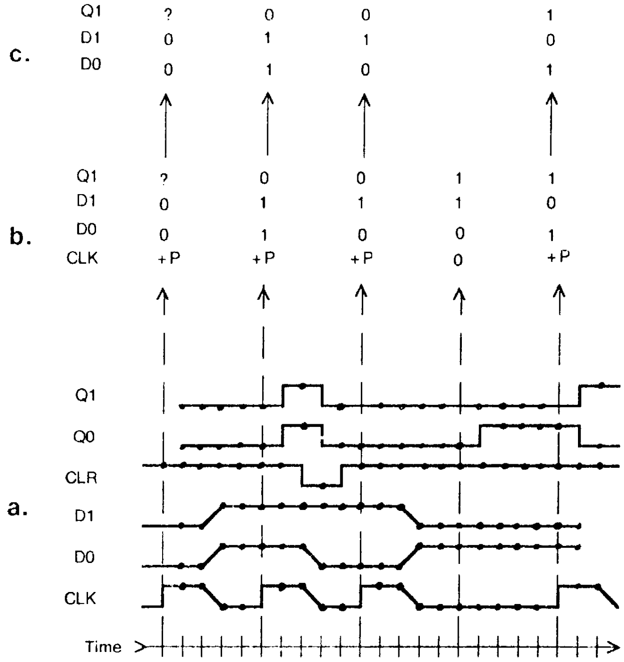

Figure 3: a,b,c: Multilevel Timing Diagrams for Device TBR

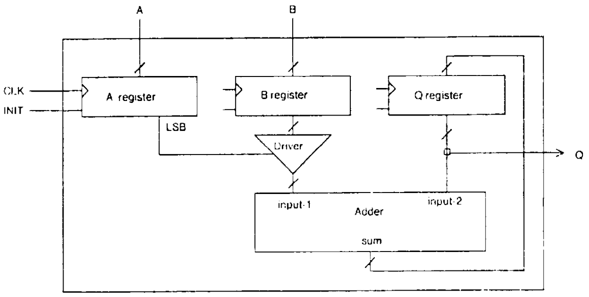

Figure 4: Sequential Multiplier with 8-bit Result Register

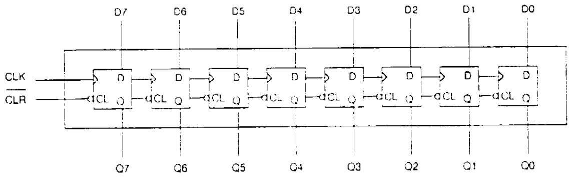

Figure 6: Eight-Bit Q Register

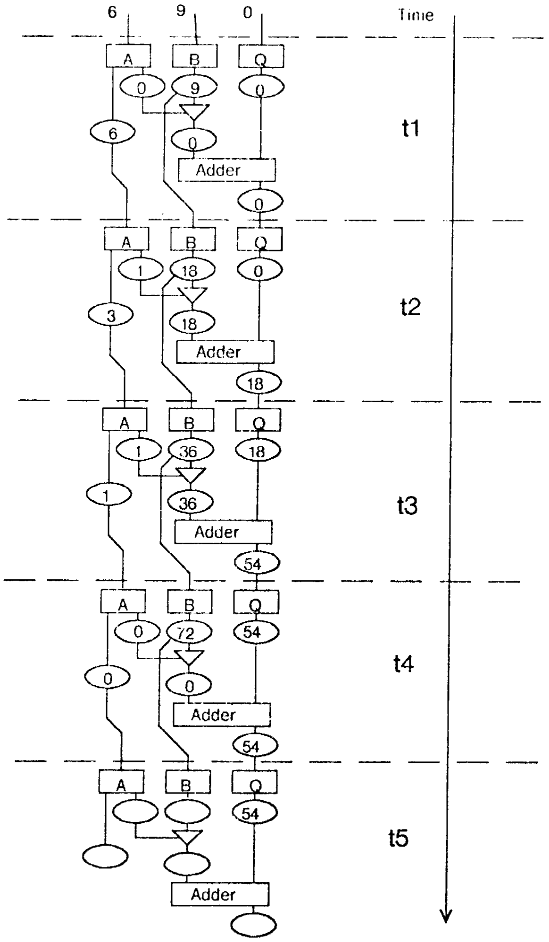

Figure 5: Multiplier Behavior Viewed Over Five Clock Cycles

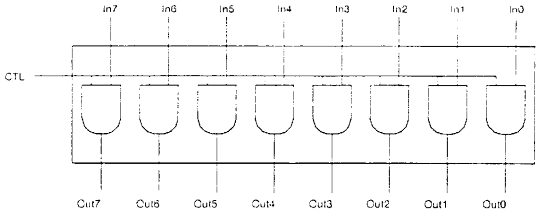

Figure 7: Eight-Bit Driver

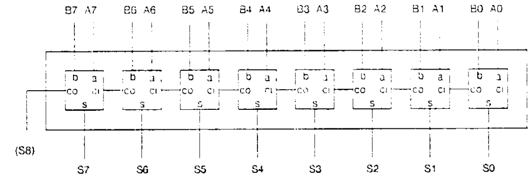

Figure 8: Eight-Bit Adder