Partitioning Circuits for Improved Testability

Sandeep N. Bhatt[1], Fan R. K. Chung[2] and Arnold L. Rosenberg[3]

[1]Dept. of Computer Science, Yale University, New Haven CT 06520.
[2]Bell Communications Research, Morristown NJ 07960.
[3]Dept. of Computer Science, Duke University, Durham NC 27706.

Exhaustive self-testing of combinational circuitry within the framework of the LSSD design discipline requires that every output node depend on a small number of input nodes. We present efficient algorithms that take an arbitrary block of combinational logic and add to it the smallest number of bits of new LSSD registers necessary to: (1) partition the logic so that no output depends on more than k inputs, and (2) maintain timing within the block (so that all input-to-output paths encounter the same number of register bits). Our partitioning algorithms conform to two different design constraints. We also show that the unconstrained partitioning problem is NP-complete.

1 Introduction

Integrated circuit technology, most particularly VLSI, has rendered the problem of circuit testing more difficult in at least three respects:

- *Scale:* VLSI circuits have tens or even hundreds of thousands of devices, compared to the few hundreds of devices of earlier technologies.
- *Access:* Components on a chip can generally be probed only from a small number of pins along the peripheries, so that most devices cannot be directly accessed.
- *Fault Models:* Perhaps worst of all, the "old reliable" *single stuck-at fault* model is of limited validity, in terms of both single faults (fault numbers increase with area and densities)

and stuck-at faults (VLSI devices have many nasty ways of failing).)

Somewhat moderating these ways in which testing has become harder are two new avenues for solving the problem. First, certain VLSI design styles, such as LSSD [5] obviate testing sequential circuits; their register-to-register design discipline allows all logic to be tested as combinational logic (after the registers have been tested). Second, the vastly increased densities of devices on chips allows one to contemplate the use of *self-testing circuitry* (STC), extra circuitry whose role is to test the other circuitry. The STC must be very small compared to the working circuitry, and so can comfortably be tested via external probes. In order for self testing to become a viable approach to the testing problem, it must be achieved within the following constraints:

- The STC must occupy a very small fraction of the chip area;
- the STC must not appreciably degrade the performance of the circuit;
- the process of testing must not be excessively time-consuming;
- the STC must give good fault coverage.

The LSSD design discipline yields an approach to self testing that satisfies these criteria in many situations. Specifically, at the cost of very little additional circuitry, one can add *linear feedback* [13] to the LSSD registers, thereby converting them in test mode to test generators [2, 10-12] which create the test inputs for each combinational logic block and to signature accumulators [1, 3, 4, 7] which collect the output results for subsequent analysis. The choice of appropriate linear feedback is crucial if one wants to be assured of good fault coverage. Indeed, in the absence of generally accepted fault models for VLSI circuits, the authors of [2, 10-12] have proposed to add linear feedback that will test the combinational logic blocks *exhaustively*.

The major shortcoming of this suggestion, as noted in [2, 10], is that exhaustive testing presupposes small "cones of influence" in the logic block to be tested; i.e., no output node of the circuit can depend on more than some small number (call it k, suggested to be about 20 in [2]) input nodes, since 2^k cycles would be required to test such an output node. The authors of [2, 10] suggest that

circuits that violate this s
the addition of new LSSI
any offending cone. It is
algorithms for performing
way, i.e., by adding the m

One overall constraint
they not impair the correc
ing of the circuit. Our sp
change the lengths of onl
shall interpret the mand
that all input-to-output p
new register.

In summary, this pape
rithms that take an arbitr
to it the smallest number

- partition the logic s
  k inputs (k being a
- maintain timing wit
  paths encounter the

We present efficient
two different design cons
discussed in Section 3, an
cussed is Section 4. Final
strained partitioning prob

2 LSSD partit

Level Sensitive Scan Desi
reduces the impossibly ha
to the very hard problem
essence of the approach
register format, with bloc
and with all feedback lo
tionally, each register is
ter for purposes of scanni
operation mode the regist
blocks of combinational lo
in an input vector and t

have many nasty ways of

which testing has become
:he problem. First, certain
obviate testing sequential
discipline allows all logic
er the registers have been
nsities of devices on chips
lf-testing circuitry (STC),
other circuitry. The STC
:king circuitry, and so can
:s. In order for self testing
:ting problem, it must be
s:

l fraction of the chip area;

grade the performance of

:cessively time-consuming;

rage.

ı approach to self testing
ıtions. Specifically, at the
ıe can add *linear feedback*
erting them in test mode
e the test inputs for each
'e accumulators [1, 3, 4, 7]
uent analysis. The choice
f one wants to be assured
ence of generally accepted
ırs of [2, 10-12] have pro-
:t the combinational logic

ɜtion, as noted in [2, 10],
all "cones of influence" in
ıt node of the circuit can
:r (call it k, suggested to
cycles would be required
rs of [2, 10] suggest that

circuits that violate this small-cone requirement be *partitioned* by the addition of new LSSD registers, so as to reduce the size of any offending cone. It is our goal in this paper to devise efficient algorithms for performing this circuit partitioning in an optimal way, i.e., by adding the minimum number of bits of new register.

One overall constraint on our partitioning algorithms is that they not impair the correctness of the circuit by upsetting the timing of the circuit. Our specific concern is that added registers not change the lengths of only some input-to-output paths. Thus we shall interpret the mandate to partition the circuit as requiring that all input-to-output paths receive the same number of bits of new register.

In summary, this paper is devoted to presenting efficient algorithms that take an arbitrary block of combinational logic and add to it the smallest number of bits of new LSSD register that will:

- partition the logic so that no output depends on more than k inputs (k being an input to the algorithm), and
- maintain timing within the block (so that all input-to-output paths encounter the same number of bits of register).

We present efficient partitioning algorithms that conform to two different design constraints: the *edge partitioning* constraint discussed in Section 3, and the *levelled partitioning* constraint discussed is Section 4. Finally, in Section 5, we show that the unconstrained partitioning problem is NP-complete.

2 LSSD partitioning

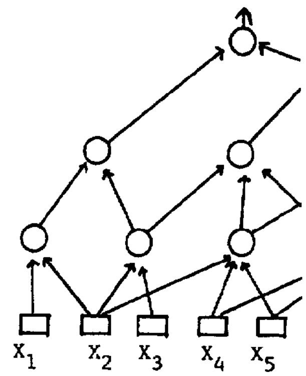
Level Sensitive Scan Design (LSSD) [5] is a design discipline that reduces the impossibly hard problem of testing sequential circuitry to the very hard problem of testing combinational circuitry. The essence of the approach is to design all circuitry in a register-to-register format, with blocks of combinational circuitry intervening, and with all feedback loops also being register-to-register. Additionally, each register is designed to be convertible to a *shift register* for purposes of scanning test inputs in and test results out. In *operation* mode the registers merely latch the signal lines between blocks of combinational logic; in *test* mode each input register scans in an input vector and transmits the vector to the combinational logic, while each output register collects the output of the logic block and scans that vector out.

This section investigates ways to partition a large combinational circuit by inserting extra LSSD registers. We illustrate some of the main concerns with an example. Figure 1a shows the skeleton of a combinational circuit; the nodes denote boolean gates, and an edge from one node to another signifies that the output of the first node is an input to the second. In general, the value computed at a node is determined by each input from which the node is reachable by a directed path. Thus, in Figure 1a, the output depends on the values in 8 input registers. Suppose that for purposes of testability we require that no node be affected by more than four registers in one clock cycle. By placing additional registers as in Figure 1b, the output (and in fact every node) is affected by at most four registers in one clock cycle; since every path from inputs 1, 2, 3, and 4 is blocked by a new register, these registers can affect the value of the output only at the next clock cycle. The modified circuit therefore meets our requirement for testability.

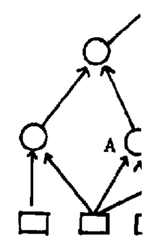
Unfortunately, the modified circuit is not functionally equivalent to the original one. This inequivalence stems from *timing* considerations: in test mode, the values computed at nodes A and B will arrive as inputs to node C at different clock cycles. As a consequence, the output of the modified circuit is affected by the initial values in the new registers, and will differ from the output of the circuit of Figure 1a.

Since we are only inserting new registers without structurally modifying the circuit, the modified circuit will be equivalent if and only if each node receives all values computed by its predecessors in the same clock cycle. At any given node this condition is met if every path from any input to the node contains an equal number of registers; since every input encounters an equal number of registers to a node, and every register contributes a delay of one clock cycle, every input to a node arrives at the same clock cycle. Figure 1c shows a circuit that is equivalent to the circuit of Figure 1a, in which every node depends on at most four registers. Although this solution uses more registers than our original attempt at a solution, one can verify that it cannot be improved on.

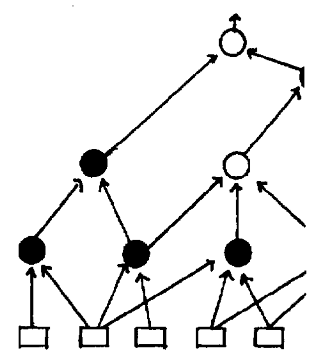
It is important to mention here that under normal operation the registers inserted for testing may be bypassed using simple circuitry. In this case one may ask whether it is at all necessary to add extra registers simply to preserve timing equivalence: why is it



(1a) An untestable circ



(1b) An in



(1c) An equivalent test

Figure 1: *Registers must b*

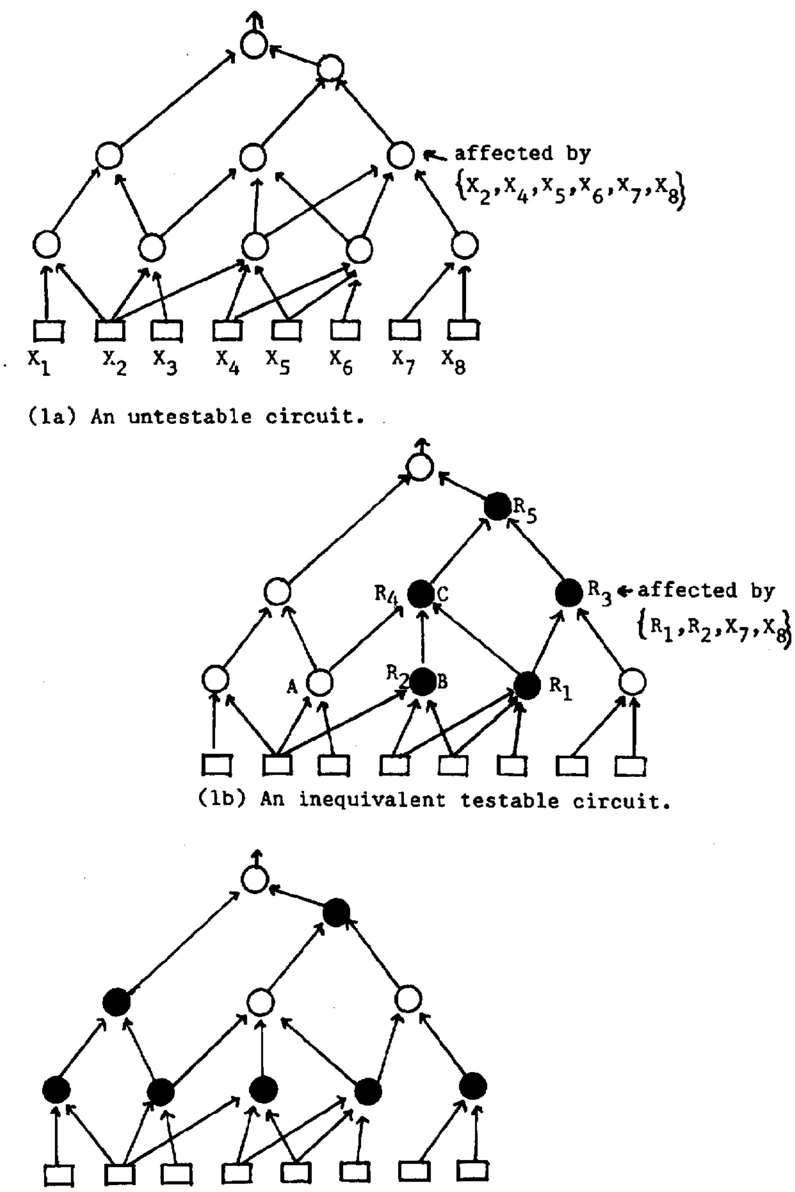
s the output of the logic

tion a large combinational We illustrate some of the : 1a shows the skeleton of ›te boolean gates, and an hat the output of the first l, the value computed at a hich the node is reachable he output depends on the for purposes of testability ıore than four registers in gisters as in Figure 1b, the d by at most four registers n inputs 1, 2, 3, and 4 is can affect the value of the modified circuit therefore

is not functionally equivalence stems from *timing* computed at nodes A and fferent clock cycles. As a circuit is affected by the ill differ from the output

ters without structurally t will be equivalent if and puted by its predecessors de this condition is met if ntains an equal number of equal number of registers a delay of one clock cycle, ne clock cycle. Figure 1c circuit of Figure 1a, in r registers. Although this inal attempt at a solution, d on.

under normal operation bypassed using simple r it is at all necessary to ing equivalence: why is it



(1a) An untestable circuit.

(1b) An inequivalent testable circuit.

(1c) An equivalent testable circuit

Figure 1: ***Registers must be placed in a way to preserve circuit timing.***

critical that the circuit compute the same function under both test and normal modes? To counter this argument, observe that the bypass circuitry will upset the timing of the circuit during normal operation; consequently, we need to insert equal delays (which may be introduced by means other than extra register bits) along every input-output path. Our intent is to focus on both the added delay and the extra hardware, hence our simplifying assumption that the delays are added by extra register bits.

In the example above, we placed registers on the nodes and not on the edges of the input circuit. When each node denotes a boolean gate which computes a single value that is sent to other nodes, this is only reasonable, since there is no need to store the same value more than once. A node may, however, also be used to represent a combinational circuit module, an adder for example, whose output values are not all the same. In this case one register cell must be placed on each outgoing edge of a node, because each edge carries a different computed value. We distinguish between these two models: in the *node partitioning* model registers are placed on nodes, whereas in the *edge partitioning* model registers are placed on edges.

In practice, many circuits are designed so that nodes fall into distinct levels, with edges directed only between consecutive levels. The FFT network is an important example of such a *levelled* circuit. In partitioning levelled circuits, a designer might require (for instance, to preserve synchronization) that for any given level, registers be placed either on every node on that level or on none of them. We call such a node partition a *levelled* partition. When a levelled circuit is symmetric in the way that the FFT circuit is, optimal node partitions are automatically levelled partitions. As a final note, observe that levelled partitions automatically maintain timing equivalence.

In summary, the two quantities that we wish to minimize are:

- the total number of bits of register added to the circuit, and
- the delay introduced, as measured by the number of registers along any path from an input to an output (any input and any output will do, since the modified circuit maintains timing equivalence).

3 Computing ‹

This section presents an e
edge partitions for circuits
at each node is no greate
not obey this fan-in/fan-o
result of one computation
one would expect the fan
since it is costly to drive
partitioning model, a nod
many common circuit mo
example, meet this fan-in/
systems whose input and

A combinational circui
Nodes of indegree 0 are
outdegree 0 are the output
reside in registers, and t
node is loaded into a reg
edges of a directed acyclic
number of registers along
$\phi(G)$ is the same. The *de*
the number of registers al

We say that (the com
u if there is a directed pa
well-timed assignment of
which there is a directed
any intermediate registers
same clock cycle. The *re*
timed circuit is the numb
a clock cycle.

The optimal edge par
follows:

Input: A directed acyclic
The indegree of each nod
than the outdegree of *v*, a
node.

Output: A well-timed a
uses the minimum numb
delay, and for which ever
no greater than *k*.

ne function under both test
ırgument, observe that the
f the circuit during normal
ert equal delays (which may
a register bits) along every
us on both the added delay
lifying assumption that the

egisters on the nodes and
When each node denotes a
value that is sent to other
ere is no need to store the
ıay, however, also be used
odule, an adder for exam-
ıe same. In this case one
tgoing edge of a node, be-
uted value. We distinguish
partitioning model regis-
ıe *edge partitioning* model

ıed so that nodes fall into
between consecutive lev-
example of such a *levelled*
, a designer might require
n) that for any given level,
on that level or on none
a *levelled* partition. When
y that the FFT circuit is,
ly levelled partitions. As a
ns automatically maintain

we wish to minimize are:

added to the circuit, and

by the number of registers
output (any input and any
l circuit maintains timing

3 Computing optimal edge partitions

This section presents an efficient algorithm for computing optimal edge partitions for circuits satisfying the property that the fan-out at each node is no greater than the fan-in. Certain circuits may not obey this fan-in/fan-out constraint, as, for instance, when the result of one computation is used repeatedly. In general, however, one would expect the fan-out of a node to be reasonably small, since it is costly to drive a signal across many wires. In the edge partitioning model, a node is used to denote a circuit module and many common circuit modules, such as adders and multipliers for example, meet this fan-in/fan-out restriction, as do clocked closed systems whose input and output ports are the same.

A combinational circuit is modeled as a directed acyclic graph. Nodes of indegree 0 are the inputs to the circuit, and nodes of outdegree 0 are the outputs. We assume that circuit inputs initially reside in registers, and that the value computed at each output node is loaded into a register. An assignment ϕ of registers to edges of a directed acyclic graph G is said to be *well-timed* if the number of registers along every path from an input to an output of $\phi(G)$ is the same. The *delay* of a well-timed assignment ϕ equals the number of registers along any input-output path of $\phi(G)$.

We say that (the computation of) a node v *depends* on a node u if there is a directed path from u to v in the circuit. If ϕ is a well-timed assignment of registers to G, and R is a register from which there is a directed path to a node v which does not contain any intermediate registers, then v *depends on register* R within the same clock cycle. The *register dependency* of a node v in a well-timed circuit is the number of registers that v depends on within a clock cycle.

The optimal edge partitioning problem is precisely stated as follows:

Input: A directed acyclic graph G and an integer k.
The indegree of each node v in G (except inputs) is no smaller than the outdegree of v, and k is greater than the indegree of every node.

Output: A well-timed assignment ϕ of registers to edges which uses the minimum number of registers and introduces minimum delay, and for which every node in $\phi(G)$ has register-dependency no greater than k.

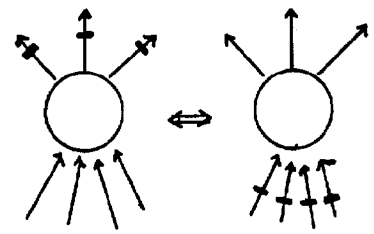


Figure 2: *The retiming operation.*

Two simple observations underlie the algorithm. The first is the use of the *retiming* operation to maintain timing equivalence while moving registers around in the circuit. Suppose that every outgoing (incoming) edge incident to a node has a register on it. By removing these registers and placing one on every incoming (outgoing) edge incident to that node as in Figure 2, the resulting circuit remains well-timed although the retimed node computes its result one clock cycle later (earlier). Retiming has been extensively used earlier in optimizing synchronous circuitry [8, 9]. Retiming is a general operation in the following sense.

Proposition 1. *Suppose that ϕ_1 and ϕ_2 are two well-timed assignments of registers to a graph G such that $\phi_1(G)$ and $\phi_2(G)$ have equal delay. Starting with the assignment ϕ_1 it is possible to obtain the assignment ϕ_2 using a sequence of retiming operations.*

The second observation exploits the fan-in/fan-out restriction, which guarantees that each application of the retiming rule which pushes registers forward (from incoming to outgoing edges) cannot increase the total number of registers in the circuit.

Proposition 2. *Suppose that a well-timed assignment $\phi_1(G)$ is obtained from $\phi_2(G)$ by a sequence of retiming operations, in each of which registers are pushed from incoming edges to outgoing edges. Then if G satisfies the fan-in/fan-out restriction (the fan-out at every node is no bigger than the fan-in), the circuit $\phi_1(G)$ contains no more registers than does $\phi_2(G)$.*

Proposition 2 gives a s
tition for graphs with the
consider a directed acyclic
in that its register-depend
are *good*, i.e., have register
ing a register on every inc
the output node becomes
edges (and hence its regist
propositions 1 and 2, this
minimizing the delay and
if a node is good, there is
Algorithm I generalize
bad nodes and having m
in stages. In each stage,
each node, to determine w
adding unit-delay we retin
be delayed by one cycle are
to the next stage using t
stage as the new inputs
terminates when each out
node in the circuit is good

To see that the algorit
observe that in the origin
bad node each of whose p
perform its computation
every node that is minim
must be retimed so that
this were not the case, th
at the start of the second
that every node that is m
stage must be retimed to l
delay must be at least as
stage adds unit delay, the

Next we argue that the
partition. Consider the n
inputs and the minimally
registers only as far back
the number of registers
the minimum required.
of the circuit that lie bet
nodes, we can prove that

ration.

ıe algorithm. The first is
ıintain timing equivalence
rcuit. Suppose that every
node has a register on it.
g one on every incoming
in Figure 2, the resulting
etimed node computes its
ming has been extensively
ircuitry [8, 9]. Retiming is
e.

ϕ_2 are two well-timed as-
hat $\phi_1(G)$ and $\phi_2(G)$ have
ϕ_1 it is possible to obtain
timing operations.

fan-in/fan-out restriction,
of the retiming rule which
to outgoing edges) cannot
the circuit.

med assignment $\phi_1(G)$ is
iming operations, in each
g edges to outgoing edges.
striction (the fan-out at
he circuit $\phi_1(G)$ contains

Proposition 2 gives a strategy to compute an optimal edge partition for graphs with the fan-in/fan-out property. For example, consider a directed acyclic graph whose single output node is *bad,* in that its register-dependency is greater than k, but all other nodes are *good,* i.e., have register-dependency no greater than k. By placing a register on every incoming edge incident to the output node, the output node becomes good because the number of incoming edges (and hence its register dependency) is no greater than k. By propositions 1 and 2, this assignment of registers is optimal both in minimizing the delay and the total number of registers. In general, if a node is good, there is no advantage in retiming it.

Algorithm I generalizes this simple idea to graphs with many bad nodes and having multiple outputs. The algorithm proceeds in stages. In each stage, we compute the register-dependency of each node, to determine whether the node is good or bad; next, by adding unit-delay we retime the circuit so that nodes which should be delayed by one cycle are converted to good nodes. Then, proceed to the next stage using the registers introduced in the previous stage as the new inputs to a truncated circuit. The algorithm terminates when each output is good, which also means that every node in the circuit is good.

To see that the algorithm computes a minimum-delay partition, observe that in the original circuit every minimally bad node (a bad node each of whose predecessors is good) must be retimed to perform its computation at least one clock cycle later. Similarly, every node that is minimally bad at the start of the second stage must be retimed so that it lags by at least two clock cycles — if this were not the case, then the node would not be minimally bad at the start of the second stage. By induction, it may be proved that every node that is minimally bad at the beginning of the ith. stage must be retimed to lag by at least i. In other words, the total delay must be at least as large as the number of stages; since each stage adds unit delay, the total added delay is minimum.

Next we argue that the algorithm computes a minimum-register partition. Consider the nodes between (and including) the original inputs and the minimally bad nodes. Since the algorithm pushes registers only as far back as necessary, Proposition 2 implies that the number of registers used within this portion of the circuit is the minimum required. By extending the argument to portions of the circuit that lie between successive stages of minimally bad nodes, we can prove that the total number of registers used is the

ALGORITHM I

Input: The adjacency list of a directed acyclic graph $G = (V, E)$ and an integer k. The indegree of each node v in G (except inputs) is no smaller than the outdegree of v, and k is greater than the indegree of every node.

Output: A minimum-delay well-timed assignment of registers to edges which uses the minimum number of registers, and for which every node has register-dependency no greater than k.

BEGIN

1. Initialize $H := V$.
2. Compute the transitive closure T of the adjacency matrix of G.
3. Compute the register-dependency $R(v)$ of each node v.
   If I is the set of inputs, $R(v) := \sum_{i \in I} T(i, v)$.
4. Compute the set V_g of good nodes, and the set M of minimally bad nodes. For each node $w \in M$, if $(u, w) \in E$ and $u \in V_g$, then place a register on the edge (u, w). For each node $w \in M$, compute the number $D(w)$ of registers placed on edges coming into w. If an output node is good but there are bad nodes remaining, place a register on the outgoing edges of the good output.
5. Remove the set of good nodes from G, (set $H := H - V_g$). If H is empty then halt.
6. Recompute the register dependency $R(v)$ of each vertex v in the new graph as follows:
   6.1. initialize $R(v) := 0$
   6.2. for each $w \in M$, if $T(w, v) = 1$ then update $R(v) := R(v) + D(w)$.
7. Go to step 4.

END

minimum required.

Finally, for circuits wit mal partitioning in $O(dN$ delay required to make ea sitive closure in step 2 tal edges is $O(N)$. Once the t register-dependencies and computed in time $O(N^2)$ d, the overall running tim may be as high as $O(N^3)$ delay d will be small, and $O(N^2)$. We leave open the be improved to always ru

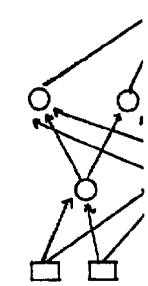


Figure 3

4 Optimal lev

In a levelled circuit, the r are directed from nodes i not "jump" levels or re is *not* required to satisj ɬ I

ɣclic graph $G = (V, E)$ and an
ʔ (except inputs) is no smaller
ɩn the indegree of every node.

ɜignment of registers to edges
ɜrs, and for which every node

the adjacency matrix of G.

v) of each node v.
$_l T(i, v)$.

and the set M of minimally
$(u, w) \in E$ and $u \in V_g$, then
ɾ each node $w \in M$, compute
on edges coming into w. If
ɜ bad nodes remaining, place
.e good output.

ʔ, (set $H := H - V_g$). If H is

$R(v)$ of each vertex v in the

then update $R(v) := R(v) +$

minimum required.

Finally, for circuits with N nodes, the algorithm finds an optimal partitioning in $O(dN^2)$ time, where d is the minimum overall delay required to make each output good. Computing the transitive closure in step 2 takes $O(N^2)$ time because the number of edges is $O(N)$. Once the transitive closure has been computed, the register-dependencies and the minimal bad nodes in each stage are computed in time $O(N^2)$ as well. Since the number of stages is d, the overall running time is $O(dN^2)$. While the worst-case time may be as high as $O(N^3)$, we believe that in practice the added delay d will be small, and that the running time will be closer to $O(N^2)$. We leave open the question of whether the algorithm may be improved to always run in time $O(N^2)$.

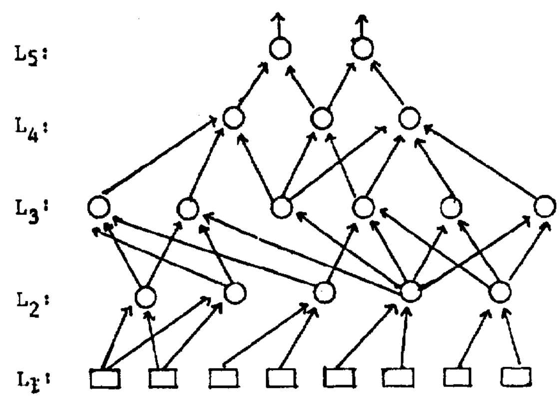


Figure 3. *A levelled circuit*

4 Optimal levelled partitions

In a levelled circuit, the nodes are divided into levels, and all edges are directed from nodes in one level to nodes in the next; edges may not "jump" levels or remain within the same level. The circuit is *not* required to satisy any other constraint, such as the fan-in/fan-out property. Figure 3 shows a levelled circuit. In a levelled partition we are required to either place registers on every node at a level or on none at all. We wish to find an optimal subset of levels on which to place registers, so that every node is good. In contrast to the previous section, this section will be concerned only with minimizing the total number of registers, not the added delay. In general there is a tradeoff between the number of registers and the added delay so that both quantities cannot be simultaneously minimized.

The simple strategy of "working forward" from minimally bad nodes will not work for levelled circuits; pushing registers further back could result in fewer registers overall. For example, considering only the first four levels of the circuit in Figure 3, although the lowest bad node is at the fourth level (assuming $k = 4$), it is better to place registers at the second level rather than the third. But when we add the fifth level, we find that it is better to place registers on the third level instead of the second and fourth levels. In short, higher levels determine which lower levels should contain registers.

Algorithm II computes an optimal partition using the standard dynamic programming technique. Suppose that we are given optimal levelled partitions for the subcircuits induced by the first level, by the first two levels, $\cdots$, and by the first $l-1$ levels. The algorithm uses these partitions to compute an optimal partition for the circuit induced by the first l levels.

We can prove by induction on the number of levels that algorithm II always gives an optimal levelled partition. For the circuit induced by levels L_1 and L_2 no registers are required; since the indegree of every node at level L_2 is no greater than k, every node at level L_2 is good. For the inductive step, suppose that the algorithm yields optimal solutions for circuits with $l-1$ or fewer levels. Consider any solution for a circuit with l levels. If any node at level L_l is bad, then the solution must place registers at level i, $(i < l)$ such that $\alpha_{l,i} \leq k$, otherwise there will still be bad nodes at level l. Since the algorithm considers all such levels i, and combines them with an optimal solution for the first i levels, the algorithm is guaranteed to find an optimal solution for the first l levels.

The running time of the algorithm is dominated by the time to compute the transitive closure of the adjacency matrix. If the graph has N nodes, then it has $O(N)$ edges, so the transitive closure can be computed in time $O(N^2)$. Given the transitive closure, the

Al

Input: The adjacency list o
and an integer k which is no
vertices in V are divided int
and output set $= L_m$.

Output: An assignment of
number of registers, and fo
no greater than k.

BEGIN

1. Compute the transitiv
2. For each level i comp
node on nodes at leve
$1 \leq j < i \leq m$, compu

$\alpha_{i,j} =$

3. For the circuit induce
contains no registers.
4. For $l > 2$, compute an
induced by levels L_1, .
   - for each level i (
which combines
   - choose a solutio
and call this sol
5. output the solution S,

END

ɜvelled circuit. In a levelled
ce registers on every node
› find an optimal subset of
ıat every node is good. In
tion will be concerned only
isters, not the added delay.
ıe number of registers and
: cannot be simultaneously

ward" from minimally bad
; pushing registers further
rall. For example, consid-
cuit in Figure 3, although
rel (assuming $k = 4$), it is
ɜvel rather than the third.
1 that it is better to place
ɜ second and fourth levels.
.ower levels should contain

ırtition using the standard
ɔse that we are given opti-
ı induced by the first level,
first $l-1$ levels. The al-
e an optimal partition for

ımber of levels that algo-
partition. For the circuit
rs are required; since the
reater than k, every node
:p, suppose that the algo-
ı with $l-1$ or fewer levels.
levels. If any node at level
egisters at level i, $(i < l)$
till be bad nodes at level
:h levels i, and combines
:st i levels, the algorithm
for the first l levels.
dominated by the time to
:ency matrix. If the graph
the transitive closure can
.e transitive closure, the

ALGORITHM II

Input: The adjacency list of a levelled directed acyclic graph $G = (V, E)$ and an integer k which is no less than the maximum indegree in G. The vertices in V are divided into levels L_i, $1 \le i \le m$, with input set $= L_1$ and output set $= L_m$.

Output: An assignment of registers to levels which uses the minimum number of registers, and for which every node has register-dependency no greater than k.

BEGIN

1. Compute the transitive closure T of the adjacency matrix of G.
2. For each level i compute the maximum dependency of a level i node on nodes at level j, $j < i$. More precisely, for each pair i, j, $1 \le j < i \le m$, compute

$$\alpha_{i,j} = \max_{v \in L_i} \left| \left\{ \sum_{u \in L_j} T(u,v) \right\} \right|$$

3. For the circuit induced by levels L_1 and L_2, the optimal solution contains no registers.
4. For $l > 2$, compute and store an optimal solution S_l for the circuit induced by levels $L_1, \ldots, L_l$ as follows:
   - for each level i $(i < l)$ if $\alpha_{l,i} \le k$, consider the solution $S_{i,l}$ which combines the solution S_i with registers at level i.
   - choose a solution from above which uses the fewest registers, and call this solution S_l.
5. output the solution S_m.

END

α's can be computed in time $O(N^2)$ in a straightforward manner. Computing an optimal solution at level l requires comparing at most $l-1$ solutions each of which is obtained by a simple calculation and a lookup, so that step 4 can be done in $O(N^2)$ time as well.

5 Untimed node partitions

In this section we show that the problem of optimally partitioning a circuit by placing registers on nodes, and without maintaining any timing equivalence is NP-complete. In other words, computing optimal partitions in untimed circuit models is intractable in general. More precisely, we consider the following problem.

Untimed Node Partition: Given a directed acyclic graph G with maximum indegree k (and a register on every input), and an integer b, is there an assignment of b or fewer additional registers to nodes of G so that every node has register-dependency at most k?

Theorem. *The Untimed Node Partition problem is NP-complete.*

Proof. The problem is trivially in NP. We reduce the 3CNF-SAT problem [6] to Untimed Node Partition. Suppose that we are given a formula ϕ over n variables $x_1, \ldots, x_n$, that contains m clauses C_i, $1 \le i \le m$, where $C_i = (x_{i_1} \vee x_{i_2} \vee x_{i_3})$. Construct the graph $G = (V, E)$ defined as follows.
The set V of vertices is:

$$\{x_i, \overline{x}_i, C_j, d_l, t_i, d'_i, d''_i \mid 1 \le i \le n,\ 1 \le j \le m, 0 \le l \le 4n-1\}.$$

The d_l's are the inputs of G. The set of edges is:

$$\{(d_{4i-1}, x_i), (d_{4i-2}, x_i), (d_{4i-3}, \overline{x}_i), (d_{4i-4}, \overline{x}_i) \mid 1 \le i \le n\}$$
$$\cup \{(x_i, C_j) \mid x_i \text{ is in } C_j, 1 \le i \le n,\ 1 \le j \le m\}$$
$$\cup \{(\overline{x}_i, C_j) \mid \overline{x}_i \text{ is in } C_j, 1 \le i \le n,\ 1 \le j \le m\}$$
$$\cup \{(x_i, t_i), (\overline{x}_i, t_i), (d'_i, t_i), (d''_i, t_i) \mid 1 \le i \le n\}.$$

Finally, to complete the reduction, set $k = 5$ and $b = n$. Figure 4 illustrates the reduction with an example. We claim that ϕ is satisfiable if and only if G can be partitioned with n registers so that each output node C_i, t_i depends on at most 5 registers. To see this, first suppose that ϕ is satisfiable. Pick a satisfying assignment,

and for each i, place a regi
$\overline{x}_i$ if x_i is false. Since at
clause node has one or n
Similarly, each t_i has a r
then the register-depender
is an instance of our probl

Next, suppose that G i
tition problem with $k = 5$
three variable nodes as pr
must be the case that one p
on it. Furthermore, each t_i
way to obtain this is by pl
$\overline{x}_i$. But this gives us a sa
QED.

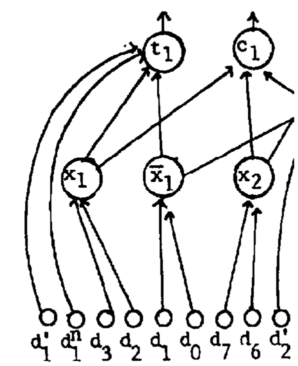


Figure 4. *Reduction f*

a straightforward manner. ɜl l requires comparing at ːned by a simple calculation ɩe in $O(N^2)$ time as well.

ːions

n of optimally partitioning and without maintaining In other words, comput- it models is intractable in ː following problem.

directed acyclic graph G er on every input), and an fewer additional registers gister-dependency at most

ı problem is NP-complete.

We reduce the 3CNF-SAT Suppose that we are given , that contains m clauses x_{i_3}). Construct the graph

$j \leq m, 0 \leq l \leq 4n-1\}$.

edges is:

$i) \mid 1 \leq i \leq n\}$

$j \leq m\}$

$j \leq m\}$

$\leq n\}$.

$k = 5$ and $b = n$. Figure .ple. We claim that ϕ is ioned with n registers so t most 5 registers. To see k a satisfying assignment,

and for each i, place a register on node x_i if x_i is true and on node $\bar{x}_i$ if x_i is false. Since at least one literal per clause is true, each clause node has one or more predecessors with a register on it. Similarly, each t_i has a register on one of its predecessors. But then the register-dependency of each output is at most 5 and so G is an instance of our problem.

Next, suppose that G is an instance of the Untimed Node Partition problem with $k = 5$ and $b = n$. Since each clause node has three variable nodes as predecessors, each in turn with 2 inputs, it must be the case that one predecessor per clause node has a register on it. Furthermore, each t_i has register-dependency 5, and the only way to obtain this is by placing exactly one register on either x_i or $\bar{x}_i$. But this gives us a satisfying assignment. The result follows. QED.

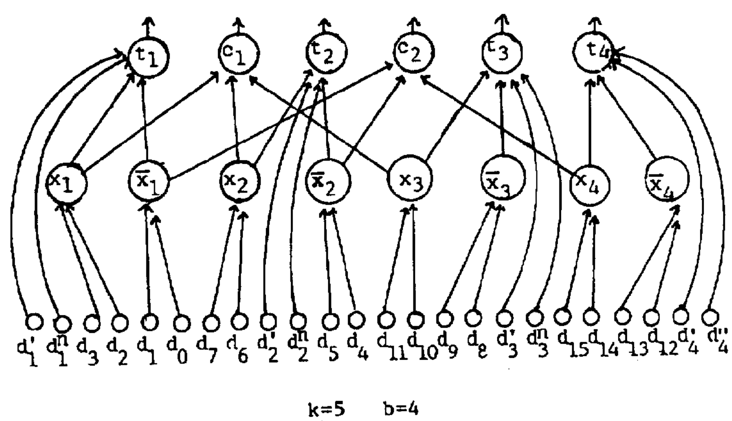


Figure 4. *Reduction for* $(x_1 \vee x_2 \vee x_3)(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_4)$

Acknowledgements

Thanks to Michael Saks of Bell Communications Research, and to Neil Immerman and Dan Gusfield of Yale University for helpful discussions. A portion of the research of the first author (SNB) and the third author (ALR) was done while visiting Bell Communications Research. A portion of the research of the third author (ALR) was supported in part by NSF Grant MCS-81-01213.

References

[1] Z. Barzilai, J.L. Carter, A.K. Chandra, and B.K. Rosen, "Diagnosis based on signature testing," IBM Report RC-9682 (1983).

[2] Z. Barzilai, D. Coppersmith, and A.L. Rosenberg, "Exhaustive bit-pattern generation, with applications to VLSI self-testing," *IEEE Trans. Comp.*, C-32, 190-194 (1983).

[3] J.L. Carter, "The theory of signature testing for VLSI," 14th ACM Symp. on Theory of Computing, 66-76 (1982).

[4] R. David, "Testing by feedback shift register," *IEEE Trans. Comp.*, C-29, 668-673 (1980).

[5] E.B. Eichelberger and T.W. Williams, "A logic design structure for LSI testability," Proc. 14th Design Automation Conf. (1977).

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, Freeman (1979).

[7] B. Konemann, J. Mucha, and G. Zwiehoff, "Built-in test for complex digital integrated circuits," *IEEE J. Solid-State and Circuits*, SC-15 (1980).

[8] C. E. Leiserson, F. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," 3rd CalTech Conf. on VLSI (1983).

[9] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI and Computer Systems*, 1 (1984).

[10] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Trans. Comp.*, C-30, 866-874 (1981).

[11] D. T. Tang and C. L. Chen, "Efficient exhaustive pattern generation for logic testing," IBM Report RC-10064 (1983).

[12] D. T. Tang and L. S. Woo, "Exhaustive test pattern generation with constant weight vectors," *IEEE Trans. Comp.*, C-32, 1145-1150 (1983).

[13] W.W. Peterson, *Error Correcting Codes*, MIT Press, Cambridge, MA. (1961).

Provably Good Patterr

Tl
Compul
Rensselа
Tro

This paper describes ɛ
random pattern test of coı
tion p of the possible pat
tor generates L patterns
that the generator generat
$pL - (pL)^2/2$. The only a
the generator are equally
gle pattern. The generatoı
feedback shift register. Th

1. Introduction.

Chip testing is becom
of total chip costs. As ci
ponents, more patterns aı
testing strategy is to calcı
each possible fault there is
correct circuit from the cir
the time required to fetch
to the circuit, severly limit
plied. Random pattern te
this limit [1]. In a randon
generates the input vecto
ates the input vectors, it c
machine cycle.

The input vectors thı
should be random. Thus