1991

On Minimizing Hardware Overhead for Exhaustive Circuit Testability

Dimitrios Kagaris
Dartmouth College

Fillia Makedon
Dartmouth College

Spyros Tragoudas
Southern Illinois University Carbondale

ON MINIMIZING HARDWARE OVERHEAD FOR EXHAUSTIVE CIRCUIT TESTABILITY

Fillia Makedon
Dimitrios Kagaris
Spyros Tragoudas

Technical Report PCS-TR91-161

EDAC'92
9/17/91

ON MINIMIZING HARDWARE OVERHEAD FOR EXHAUSTIVE CIRCUIT TESTABILITY

Dimitrios Kagaris, Fillia Makedon and Spyros Tragoudas

Dept.ofMath. &Computer Science
Bradley Hall
Dartmouth College
Hanover, NH 03755

Computer Science Department
Faner Hall
Southern Illinois University
Carbondale, IL 62901

Abstract

Exhaustive built-in self-testing is given much attention as a viable technique in the context of VLSI technology. In this paper, we present a heuristic in order to make exhaustive testing of combinational circuits practical. The goal is to place a small number of register cells on the nets of the input ciruit so that the input dependency of combinational elements in the circuit is less than a small given integer k. Our heuristic guarantees that each output can be individually tested with 2^k test patterns and can be used as a subroutine to generate efficient test patterns to test all the outputs of the circuit simultaneously. For example, we can connect the register cells in a Linear Feedback Shift Register (LFSR).

Minimizing the number of the inserted register cells reduces the hardware overhead as well as the upper bound on the number of test patterns generated. A heuristic approach has been proposed only for the case when an element in the circuit schematic denotes a boolean gate. An element may, however, also be used to represent a combinatorial circuit model. Our heuristic applies to this case as well. Extensive experimentation indicates that the proposed technique is very efficient.

Key words: Design for testability, test pattern generation, built-in self test.

Submitted to EDAC'92

Typos in the figures were corrected in the original version)

ON MINIMIZING HARDWARE OVERHEAD FOR EXHAUSTIVE CIRCUIT TESTABILITY

Dimitrios Kagaris, Fillia Makedon and Spyros Tragoudas

Dept.of Math. &Computer Science
Bradley Hall
Dartmouth College
Hanover, NH 03755

Computer Science Department
Faner Hall
Southern Illinois University
Carbondale, IL 62901

1 Introduction

The problem of circuit testing has received new dimensions in the VLSI technology since a very large number of components need to be tested. The method to deal with the problem - motivated by the very VLSI style itself- is to incorporate some additional circuitry in the original circuit to do the testing of the latter. In this "built-in self-testing" approach the problem is to minimize the testing circuitry overhead as well as the time of the actual testing process.

There have been suggested two methods for built-in self-test : random and exhaustive testing. The problems of random and the potential benefits of the exhaustive testing [2] make the latter method more desirable. Our work in this paper contributes to that direction.

A VLSI design style has been proposed [1] that reduces the very hard problem of testing a single sequential circuit to the less hard problem of testing a number of combinational circuits. If an output of the combinational circuit depends on a number i of circuit inputs, then at least 2^i patterns must be generated for the exhaustive test of the circuit, which may well be prohibitive. The general method that has been proposed [2, 3, 4] to tackle with the problem is to introduce additional inputs to the circuit, so that each element is dependent on a small number k of the additional and original inputs. (In practice k is allowed to be at most 20.)

The additional inputs, which will be referred to as "testers", are register cells. These cells, ultimately to be connected to one or more LFSRs for test pattern generation, are generally placed on the connecting wires between the elements. In the case when the elements are boolean gates whose (single) output feeds several other nodes, one has the option of placing one cell on the element instead of placing a number of cells on some of the outgoing wires. These two placement strategies are referred to as the *node–blocking* and the *edge–blocking* model, respectively. In [7] we have shown the following two theorems:

Theorem 1.1 *Every feasible solution under the node–blocking model is a feasible solution under the edge–blocking model.*

Theorem 1.2 *The minimum number of testers under the edge–blocking model is less than or equal to the minimum number of testers under the node–blocking model.*

The second theorem is a consequence of the circuit representation we adopt. A combinational circuit corresponds directly to a hypergraph, and, by some transformation, to a directed acyclic graph. For example, a circuit in which gate 1 feeds gates 2,3 and 4, may be depicted as the hypergraph of Figure 1a, or as the graphs in Figures 1b and 1c. The point is that instead of working under the node–blocking model on a graph like that of Figure 1b, we work under the edge–blocking model on a graph like that of Figure 1c. Thus, we gain in applicability, since the node–blocking model is not valid in levels other than the gate level.

In the directed acyclic graph we distinguish nodes with indegree 0, which are the inputs of the circuit, and nodes with outdegree 0, which are the outputs. We assume every edge is initially labelled by 0 except for the outgoing edges of every input node i which are labelled by $i, 1 \leq i \leq n$, where n is the number of inputs to the circuit. A label a affects node u if there is a path from an input node to u and a is the last non-zero label of an edge on the path. For each node u we define $Aset(u)$ to consist of all labels affecting node u. We want to relabel the smallest number of 0-labelled edges by new and unique additional labels greater than m so that the size of the $Aset$ of any node does not exceed a given limit k. For a given k a node is called *divergent*, if $|Aset(u)| > k$. If there is no labelling of the edges of the graph such that u can be made non-divergent, the problem is infeasible.

An example circuit graph is shown in Figure 2. The value of k is five. If the divergent nodes are corrected locally (as in the figure), we need to place three testers. However, there is a feasible solution with two testers.

Both (node and edge) models have been shown to be NP-hard [3, 6]. Bhatt et al. present algorithms for special cases of circuits . Jone and Papachristou [4] present a heuristic for general circuits using the node model (although due to often occuring infeasible solutions they allow placement of some testers on the edges as well.) In [7], we have obtained solutions for special cases of circuits like multistage graphs and series-parallel graphs. The heuristic presented here is the first one using the edge–blocking model.

This heuristic forms the basis of a framework for testability [8]. In the first stage of this framework, we introduce a low hardware cost and in a second stage, by means of an indicative metric and hopefully small additional hardware overhead, we derive an efficient test pattern generation process for the exhaustive testing of the whole circuit. (The second phase is centered around the primitive polynomial technique of Barzilai et al. [5].)

The structure of the paper is as follows : In Section 2 we present the basic scheme of the heuristic and in Section 3 the specific version we propose. In Section 4 we describe several other versions that we implemented. Finally, in Section 5 we discuss our experimental results and the difficulty of obtaining optimal solutions.

2 The heuristic - Basic scheme

Our technique proceeds in fronts through the graph in a breadth-first search manner. A *front* consists of all divergent nodes which have no divergent parents. We also keep track of the set of nodes from which we start the search for the next front of divergent nodes. This set, referred to as the *boundary*, is not necessarily the last front encountered. (Initially, it is the set of the input nodes.) In fact, the procedure for the boundary formation is one of the options of the heuristic.

At each front encountered, we try to estimate the costs of several ways of proceeding further. One way is to locally (greedily) correct the divergent nodes themselves, i.e., place testers on their incoming edges so that they become non-divergent. Another way is to choose a subset of the edges between the current boundary and front (e.g., find the edges that comprise a minimum cut between the boundary and the front) and place testers on them. In this way, the current divergent nodes may not even be all corrected, but this provides a way to escape from local minima.

A lookahead metric determines at each step the candidate tester placement that will be preferred. In all versions, this metric is the number of testers required to correct the rest of the circuit (i.e., below the current front) using the greedy local correction for any subsequently encountered divergent node. The placement giving the lowest such cost is preferred and the algorithm proceeds iteratively to a new front until the whole circuit is corrected. Once we have obtained a feasible configuration of testers (list of edges that are to be blocked by testers), we derive a new partial configuration from the old one by a perturbation method to be specified, and run the heuristic all over again starting from the boundary of the input nodes. At this point, however, we take into account this imposed list of testers. (The first run of the heuristic can be viewed as performed with an empty such list.) Thus, we develop a "neighborhood" of c feasible configurations for some specified size c and take as one candidate solution the configuration with the minimun number of testers in the neighborhood. The whole process is repeated a number of times r. Each repetition starts with a randomly selected part of the last candidate solution as the new imposed list of testers. The best among the r candidates is the final solution. The generic schematic is as follows :

```
0. Start with an empty list of testers $T$.
FOR a specified number of times $r$ DO
    FOR a specified number of times $c$ DO
        1. Set boundary $B$ to contain the input nodes of the circuit.
        2. Find the next front $F$ of divergent nodes starting from $B$ and taking into account $T$.
        3. IF no divergent node was found, THEN go to 7.
        4a. Let $t1$ be the number of testers given by the local correction strategy.
            Let $T1$ be the resulting new list of testers ($|T1| = |T| + t1$).
            Also let $B1$ be the new boundary.
        4b. Let $t2$ be the number of testers given by the global correction strategy.
            Let $T2$ be the resulting new list of testers ($|T2| = |T| + t2$).
            Also let $B2$ be the new boundary.
        5a. Assuming list $T1$ is in effect, find the number $l1$ of testers required to correct
            all subsequent divergent nodes using the local correction routine only.
        5b. Assuming list $T2$ is in effect,find the number $l2$ of testers required to correct
            all subsequent divergent nodes using the local correction routine only.
        6. IF $t1 + l1 \leq t2 + l2$THEN set $T$ to $T1$, $B$ to $B1$ and go back to 2.
                                   ELSE set $T$ to $T2$, $B$ to $B2$ and go back to 2.
        7. Set $T_i$ to $T$ and apply a perturbation procedure in order to obtain
           a list of testers $TB$ from $T$, with $|TB| < |T|$.
        8. Set $T$ to $TB$ .
    END FOR
    9. Let $T_j$ be the minimum sized $T_i, 1 \leq i \leq c$.
    10. Set $T$ to a random subset of $T_j$.
END FOR
11. The minimum sized $T_j, 1 \leq j \leq r$ is the result.
```

3 The Proposed Heuristic- Details

The basic scheme above allows for many alternative implementations depending on the choice of the routines for local correction, global correction, perturbation, boundary formation etc. In this section, we describe in some detail the version of the heuristic that we found behaving best, and in the following section we present other competitive versions.

The strategy we use to locally correct a divergent node inspects all its incoming edges in descending order of *Aset* size, placing a tester on each edge until the node becomes non-divergent. This is a fast, although not optimal way of placing a small number of testers.

As a global correction strategy, we use the max-flow min-cut algorithm with unit edge capacities to find a minimum directed cut between the current boundary and front. This is a way of placing a small number of testers in hope to reduce the size of subsequent fronts. We also compute the costs for placing testers on the edges of all subsequent minimum cuts between the first mininum cut and the current front. The intuition for the latter is that a placement of testers closer to the front makes the generation of new divergent nodes above the front less likely.

Before comparing the costs of the alternative placements, we perform a filtering on the testers of each candidate set by replacing all k testers on the outgoing edges of a node with outdegree k and indegree one with a tester on its single incoming edge.

Theorem 3.1 *[7] An optimal solution obtained under the restriction that no 1-indegree node can have a tester on its incoming edge, is no better than an optimal solution with no such restriction.*

The creation of the neighborhood around the initially derived solution is done as follows: For each node which is the beginning of an edge in the list of blocked edges we compute the number f of its non-blocked incoming edges and the number b of its blocked outgoing edges. If $f < b$, we remove the testers from all outgoing edges and block all of the remaining unblocked incoming edges.If there are no directly generated divergent nodes, we update the list of blocked edges accordingly, and iteratively arrive at the final list to be used in the next run of the heuristic.

The time complexity of the heuristic is determined by the max–flow min–cut routine. Assuming bounded degree nodes—which is often the case in VLSI— this routine is called no more than m times, where n and $m = O(n)$ is the number of nodes and the number of edges in the circuit graph.

Theorem 3.2 *[7] The time complexity of the heuristic is $O(n^4)$.*

However, in practice the the number of subsequent cuts is a constant and therefore the time complexity is $\mathrm{O}(n^3)$.

4 Versions

We have programmed several variations and extensions of the basic scheme. These were introduced for comparison reasons as well as in view of their merit in the subsequent stages of the framework. A mnemonic for each version is given inside parentheses :

- (**L**) Perform the local correction of a divergent node by finding the least expensive subset of incoming edges to be blocked among all 2^k possible subsets, where k is the indegree of the node. This makes the local correction to be a real local optimum.

- (**B**) Obtain the minimum cut by assigning a capacity $n - |Aset(u)|$ to each edge (u, v), where n is the number of circuit inputs. This blocks edges through which many affectors pass.

- (**S**) Obtain the minimum cut by assigning a capacity $p - |Aset(u)|$ to each edge (u, v), where p is the size of the largest $Aset$ encountered between the boundary and the front. The intuition is the same as for **B**, but smaller capacities mean faster running times. The results may be different though.

- (**R**) Obtain the minimum cut by assigning a capacity $|Aset(u)|$ to each edge (u, v), where p is the size of the largest $Aset$ encounterd between the boundary and the front. Although the capacity assignment is contrary to the intuition for **B**, this may result in a more efficient test pattern generation routine.

- (**M**) Consider only a predifined number M of succesive minimum cuts as alternative candidate placements at each front, instead of all succesive minimum cuts. This would reduce the running time (although the results may be different).

- (**D**) Obtain the cut by making its sources be all the divergent nodes met so far and not only those in the previous front. This makes the non-local alternative to be more global.

- (**C**) Consider as a single alternative placement the outgoing edges of any non-divergent node between the current boundary and front whose $Aset$ size approaches k and its outdegree is relatively large. This is a good precautionary measure since these nodes are most likely to be the cause of divergencies below.

- (**H**) Consider as a single alternative placement the h heaviest edges encountered between the boundary and the front, where the weight of an edge (u, v) is taken to be $|Aset(u)|$. (The edges considered should not be adjacent.) The intuition is the same as in **B**, but there are no time–consuming flow techniques involved.

- (**P**) In this version, the goal is to place testers so that the circuit is partioned into segments.(Nodes in two different segments never have overlapping *Asets*.) The segmentation is generally desirable for efficient test pattern generation. This goal is achieved by two means. As a local correction routine, we enforce all incoming edges of a divergent node to be blocked by testers. As a global correction routine, we apply the undirected max–flow min–cut algorithm, which guarantees that the two parts of the cut have no common affectors. Another reason for considering this version is to obtain a technique similar to the one used under the node–blocking model by [4].

- (**G**) Obtain a neighborhood by each time unblocking any outgoing edges of a node with indegree one and blocking its incoming edge in case no immediate divergent nodes are generated. This examines the effects of a lighter perturbation method than the one in the proposed version.

5 Experimental Results

We ran all the above versions on directed acyclic graphs generated randomly by specifying the required number of inputs, the maximum indegree and outdegree, and a lower bound on the number of nodes and outputs in the graph. A small sample of the results is given in Table 1. The results of the proposed heuristic are given under column **E**. For this column, the number in parentheses is the corresponding percentage of blocked edges. The e/r label indicates whether the preferred solution was obtained from the original empty list of testers or from a randomly imposed one. The other columns give the results obtained by changing a part of the heuristic according to the indicated version. (In **M** only one (unit-capacity) cut was allowed.)

Table 1

No	E			R	S	B	M	C	L	P
1	5	(3.45)	(e)	5	7	5	6	5	6	35
2	4	(7.62)	(e)	4	4	4	4	4	4	4
3	14	(26.92)	(e)	14	14	14	14	14	14	15
4	5	(25.0)	(e)	5	5	5	5	5	5	7
5	47	(8.88)	(r)	46	47	51	47	47	49	113
6	53	(12.34)	(e)	53	62	64	60	50	44	170
7	350	(34.44)	(e)	351	359	386	378	370	352	984
8	213	(20.07)	(e)	212	275	261	286	240	222	457
9	90	(17.03)	(r)	92	98	101	90	95	94	189
10	151	(14.35)	(e)	151	153	146	151	151	155	318
11	229	(19.88)	(r)	240	250	249	242	233	226	483
12	348	(31.67)	(e)	354	335	354	341	336	350	759
13	112	(20.28)	(r)	112	116	150	150	117	113	277
14	45	(14.52)	(e)	46	50	51	54	42	44	124

The maximum graph size was 1000 nodes. The running time on a SUN 3/60 for graphs of this size and for $k = 20$ was less than an hour, which is acceptable due to the nature of the application. The results in rows 2 and 3 are for the real circuit of Figure 5, for a k of three and four respectively. The results in row 4 are for the real circuit in Figure 4, for a k of three. (Both circuits are from the Research Triangle Institute.) As far as the comparison of our heuristic with the optimal is concerned, we were able to obtain results only for small graphs. In the first four instances above this optimal was achieved. Unfortunately, the optimal solution for any other instance (even the relatively small ones) could not be calculated despite the use of a CRAY X-MP (UT System Computing Center, Austin, Texas). The difficulty in finding any exhaustive method other than the trivial one is attested be the following theorem :

Theorem 5.1 *A feasible solution with t testers does not necessarily imply that there is a feasible solution with t+1 testers. Furthermore, this is true independently of the relative ordering of the values of the allowable limit k, the maximum indegree in the graph and the number of testers already present.*

An example is shown in Figure 3, where one tester has been placed in order to satisfy the limit $k = 2$, but there is no place for a second tester. This precludes even slight improvements by using techniques like binary search in order to find a better solution from a given feasible one.

As mentioned in the Introduction, this is the first heuristic for the edge–blocking model. The work in [4] is the most relative to this one, but the model used there is a combination of edge- and node-blocking models and applies to gate level only. However, we were not able to have either the program or some of the input instances in [4] in order to give comparative results. Among our versions, version **P** seems to be more similar to that heuristic.

References

[1] E. B. Eichelberger, T. W.Williams, "A Logic Design Structure for LSI Testability", *ACM/IEEE Design Automation Conference* , vol. 14, pages 462–467, 1977.

[2] E. J. McClauskey, "Built–in Self–testing Techniques", *IEEE Design and Test*, pp. 21–28, April 1985.

[3] S. N. Bhatt, F. R. K. Chung, A. L.Rosenberg, "Partitioning Circuits for Improved Testability", *MIT Conference on Advanced Research in VLSI* , pp. 91–106, April 1986.

[4] W. Jone and C. A. Papachristou, "A Coordinated Approach to Partitioning and Test Pattern Generation for Pseudoexhaustive Testing", *ACM/IEEE Design Automation Conference* , vol. 26, pages 525–530, 1989.

[5] Z. Barzilai, D. Coppersmith, A. L. Rosenberg. "Exhaustive Bit Generation with Application to VLSI Self–Testing", *IEEE Transactions on Computers*, vol. C-32, no. 2, pages 190–194, 1983.

[6] J. Doenhardt, "Partitioning Circuits for Improved Testability–The Computational Complexity of Various Methods", *personnal communication* ,submitted for journal publication.

[7] D. Kagaris, F. Makedon, S. Tragoudas, "Register Insertion for Pseudoexhaustive Testing", Technical Report, Department of Math. & Computer Science, Dartmouth College, August 1991.

[8] D. Kagaris, F. Makedon, S. Tragoudas, "A Framework for Testability", paper in preparation.

iii

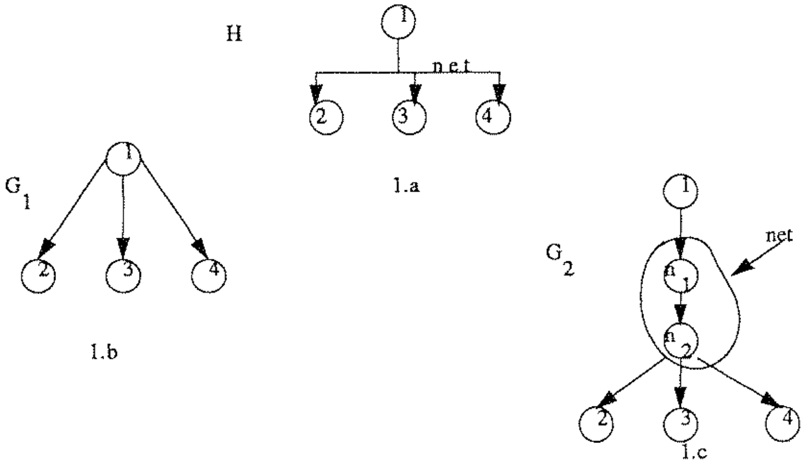


Figure 1: Directed graph transformations from the input hypergraph

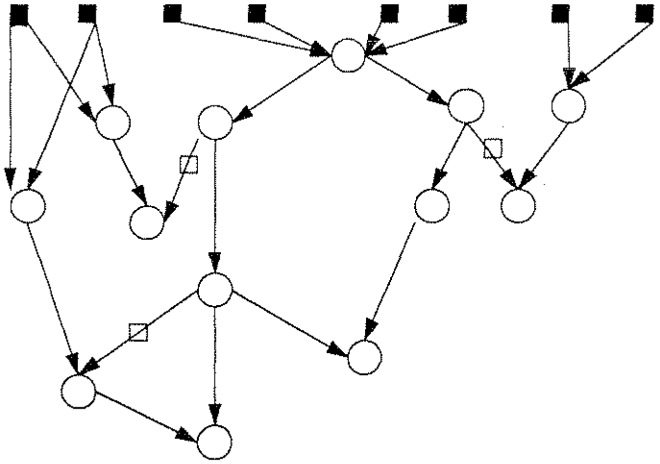


Figure 2: A feasible configuration for k = 5. The black spuare nodes indicate primary inputs. The empty spuare nodes are the inserted register cells.

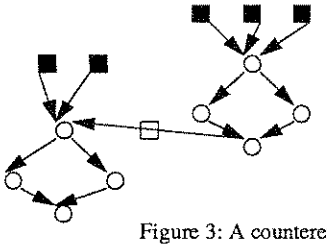
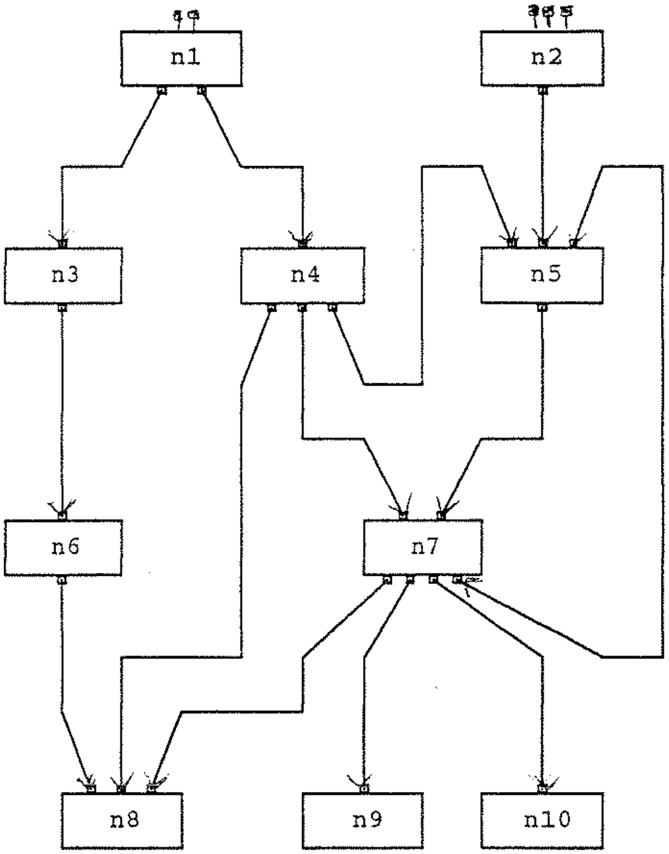


Figure 3: A counterexample.

t

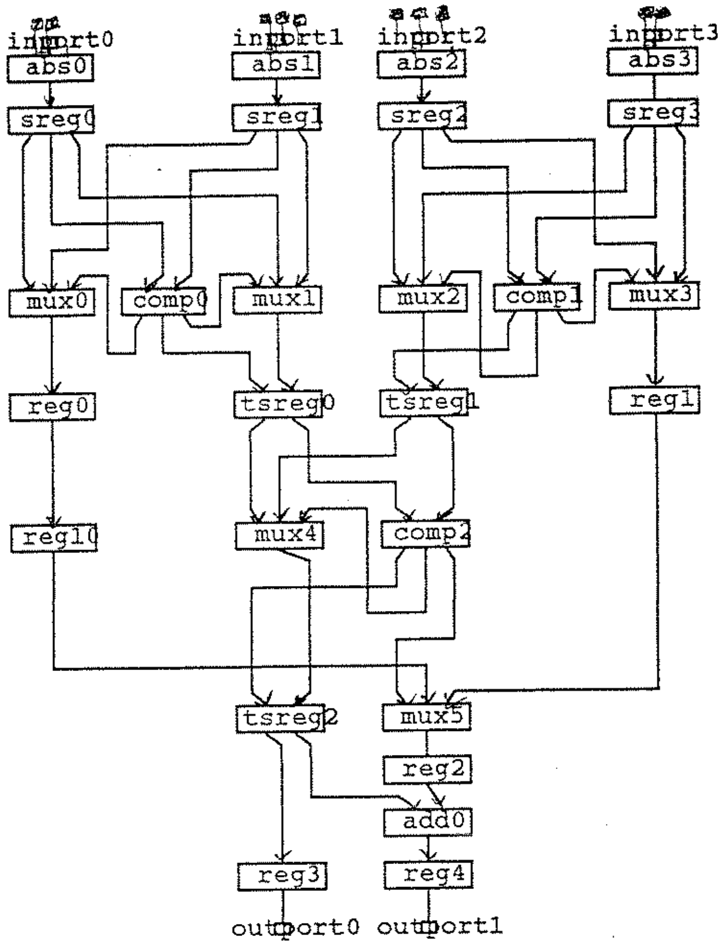
Figure 4: A schematic. The black square node indicate primary inputs.

Fiogure 5: A schematic: The black spuare nodes indicate primary inputs.



sas10.hwg

FIGURE 4



dmagproc.hwg

Figure 5