Defence Science Journal, Vol 45, No 3, July 1995, pp 255-265

New Metric-Based Algorithm for Test-Vector-Generation in VLSI Testing

M.V. Atre and V. Latha

Advanced Numerical Research & Analysis Group (ANURAG), Hyderabad-500 258

ABSTRACT

A new algorithm for test-vector-generation (TVG) for combinational circuits has been presented for testing VLSI chips. This is done by defining a suitable metric or distance, in the space of all input vectors, between a vector and a set of vectors. The test vectors are generated by suitably maximising the above distance. Two different methods of maximising the distance are suggested. Performances of the two methods for different circuits are presented and compared with the random method of TVG. It was observed that method B is superior to the other two methods. Also, method A is slightly better than method R.

INTRODUCTION

Advances in semiconductor and miniaturisation technologies have given birth to very large scale integrated (VLSI) circuits which form the basis of most of the chips in the present day computing systems. Also, most of the recent application specific integrated circuits (ASICs) fall within the VLSI category.

This has led to a proliferation of ideas, methodologies and processes in VLSI circuit design and fabrication[1]. Especially on the software front, some of the concepts involved are[2,3] : description and modelling of faults, simulation techniques for design verification, simulation of faults, test-vector-generation techniques for fault detection, etc.

Faults in circuits can be of various types : oxide and metal layer defects, contamination, contact and interconnect defects, corrosion, metal failures, etc. Many of these faults can be modelled for transistors, gates or functional levels[4]. At present, gate level fault modelling has been found to be adequate, detailed, and yet tractable, for very large circuits.

The most popular fault model in gate level simulation is the stuck-at-fault (s-a-f) model[5], where each line interconnecting any two gates can be assumed to be either stuck-at-zero (s-a-0) or stuck-at-one (s-a-1). If there are n such lines in a circuit, the total number of possible faults are 3^n–1 (including the dont-care condition). This is very large number in most cases, and hence the model is usually restricted to single s-a-fs, which are then $2n$ in number.

Having modelled a given fault, it is necessary to detect that fault by giving a suitable input pattern to the circuit. This suitable pattern is called a test-vector (TV). A TV is an input pattern which gives different outputs for the fault-free circuit and the faulty circuit, and hence detects that fault. One TV may, of course, detect more than one fault in a circuit. It is desirable to generate a set of TV's which will detect as many faults as possible. The process of generating the required TV's is called test-vector-generation (TVG)[1]. Hence, it is necessary to have a very efficient TVG package which will generate the TV's in as minimal a time as is possible.

The key ideas involving TVG are illustrated in Sec. 2. Section 3 gives the basic idea involving a new algorithm via, a metric in the space of input vectors. Two methods of TVG based on the algorithm given in

Received 2 January 1994, revised 01 August 1994

Sec. 3 are presented in Sec. 4. Section 5 describes the implementation of the new methods, as also the performance results for various circuits. Discussion and conclusions are given in Sec. 6.

2. TEST-VECTOR-GENERATION

The test-vector-generation (TVG) is one of the key issues in the design and manufacture of chips. Basically, it involves finding a set of input patterns (called test-vectors) to the circuit which gives different outputs for fault-free and faulty cases. This is accomplished by using a fault-simulator which, for a given TV, finds out all single s-a-fs which are detected by the TV out of all possible simulated faults.

Naturally, a quantity to define is the fault-coverage (f-c) given by

$$= \frac{100 \times \text{no. of faults detected}}{\text{total no. of simulated faults}} \quad (1$$

As mentioned in the introduction, the most popular fault-model, which is representative of the majority of possible physical faults after manufacture, is the s-a-f model at the gate level. The primary aim of any good TVG package is to generate vectors in minimal time giving maximal f-c.

In principle, one can always generate an exhaustive set of all possible vectors that can be fed into a circuit. For example, given a circuit with m primary inputs, the total number of all possible inputs is 2^m. But for typical pin number, like $m = 100$ (or more), the total number of vectors is 2^{100}, which is astronomical. It is thus desirable to cleverly and quickly generate a subset of all the possible vectors, which nevertheless gives as high a f-c as possible. This entails finding suitable algorithms to reduce the search space of vectors, and this is the basic philosophy behind the various TVG schemes.

The TVG philosophy can be broadly classified into two parts: global and local:

(i) The global approach aims at finding quick/heuristic means of TVG which mostly adopt random or other pseudo-random techniques to generate vectors and are usually used for reaching a f-c of about 75–85 per cent in the shortest possible time.

(ii) Generating vectors for the remaining 15–25 per cent of f-c is very difficult and requires an approach which is fully deterministic. In this local category of TVG, a particular fault is identified and a method is set up to generate a vector to catch that fault (unlike in the global case where a vector is first generated and the various faults it catches are then determined). This involves some of the well-established techniques, such as path-sensitisation, justification, back-tracking, incorporation of testability analysis, etc. Some of the standard algorithms used are D-algorithm, PODEM, etc.

The basic idea involving a new search algorithm in the global category is given in the next section.

3. NEW SEARCH ALGORITHM VIA A METRIC

In the global TVG scenario, a new vector is either generated randomly or by invoking some heuristic principle. The basic aim of the heuristic principle adopted in this paper is to find input vectors which are as far from each other as possible, i.e. to explore in the space of all input vectors, as wide an area as possible. Hopefully, this scheme will yield TV's which are representative of a very diverse set of possible TV's, thereby yielding a very high f-c for small number of vectors.

Let $(V)_J$ be the set of J number of TV's generated up to a given instant. It is then desirable to generate the next vector V_{new}, which is farthest from the given set of points representing the already generated vectors $(V)_J$. This can be quantified as follows.

Let the total number of primary inputs be m. Then there are 2^m possible combinations of inputs where each input vector V can be thought of as a binary m-tuple, e.g.,

$$V = (b_1\, b_2 \ldots\ldots b_m) \text{ with } b_i = 0 \text{ or } 1; \quad i = 1 \ldots\ldots m \quad (2)$$

Thus, the different possible inputs can be thought of as the different vertices of an m-dimensional hypercube H_m and can be represented as a set $(V)_m$ where

$$(V)_m = V_i; \quad V_i = \text{Vector of } H_m; \quad i = 1, \ldots\ldots 2^m \quad (3)$$

Let each member of $(V)_J$ has a representation as given in Eqn (2),

$$\begin{aligned} V_1 &= (b_{11}, b_{12}, \ldots\ldots b_{1m}) \\ V_i &= (b_{i1}, b_{i2}, \ldots\ldots b_{im}) \\ V_J &= (b_{J1}, b_{J2}, \ldots\ldots b_{Jm}) \end{aligned} \quad (4)$$

where each b_{ij} is either 0 or 1

We define the centre-of-mass of this set $(V)_J$ as a vector V_J^{cm} where

$$V_J^{cm} = (C_1, C_2 \quad C_m \tag{5}$$

with

$$C_i = \frac{1}{J} \sum_{j=1}^{J} b_{ji} \quad i = \quad m \tag{6}$$

Obviously $0 \leq C_i \leq 1$ *and the single vector* V_J^{cm} can be thought of as a hypothetical point within H_m and to be representative of the entire set $(V)_J$. Let the representation of the next vector under consideration be

$$V_{next} = (n_1, n_2, \ldots . n_m)$$

where $n_i = 0$ or 1; $i = 1 \ldots m$ (7)

The desired algorithm can be realised by defining a suitable metric or equivalently a distance D, in the space of input vectors (i.e., on H_m) between a vector V_{next} (i.e., a point on H_m) and a set of vectors $(V)_J$ (i.e., a set of points on H_m represented by $(V)_J$. $(V)_{new}$ is then that $(V)_{next}$ obtained by maximising the above distance D. This can be done as follows:

The distance $D\{(V)_J, V_{next}\}$ between the set of vectors $(V)_J$ and the vector V_{next} is defined as follows:

$$D\{(V)_J, V_{next}\} = d(V_J^{cm}, V_{next}) \tag{8}$$

where d is the normal Euclidean measure on H_m thought of as embedded in the m-dimensional Euclidean space R_m, i.e.,

$$d(V_J^{cm}, V_{next}) = \left\{ \sum_{i=1}^{m} (C_i - n_i)^2 \right\}^{1/2} \tag{9}$$

It is obvious that D satisfies all properties of a distance

A new vector V_{new} is chosen as that next vector V_{next} which maximises the above distance D, i.e., $D\{(V)_J, V_{next}\}$ is maximum for the given $(V)_J$.

The idea of finding a vector farthest from a set of vectors, will be used to find the test vectors as follows:

In any iteration, let $(V)_J$ be the set of vectors already generated, leading to a fault-coverage of $(\text{f-c})_J$. Choose V_{new} (as suggested above) such that it is the farthest from V_J. If V_{new} increases $(\text{f-c})_J$, then include V_{new} in $(V)_J$ to get $(V)_{J+1} = (V)_J + V_{new}$ and repeat the process. If $(\text{f-c})_J$ is not increased, then choose the next farthest vector and repeat the process.

The pseudo-code for the general scheme of TVG is given as follows:

1. $(V)_J \leftarrow$ Initial seed vectors.
2. Estimate $(\text{f-c})_J$
3. While $(\text{f-c})_J <$ (Required value)
   begin
3. Repeat
   find V_{new} such that $D\{(V)_J, V_{new}\}$ is maximum
   Until $\{V_{new} \in (V)_J\}$
3.2 Estimate $(\text{f-c})_{new}$.
3.3 If $\{(\text{f-c})_{new} > (\text{f-c})_J\}$
   begin
   $(V)_J \leftarrow (V)_J + (V)_{new}$
   $(\text{f-c})_J \leftarrow (\text{f-c})_{new}$
   end
   else
   go to step 3.1.
   end
4 $(V)_J$ is the required set of TVs.

4. DIFFERENT METHODS FOR MAXIMISING $D\{(V)_J, (V_{next})\}$

Two methods of maximising D or equivalently, for finding a vector farthest from a set of vectors, are described in the following:

4.1 Method A

Let $(V)_J$ be a given set of TV's at any instant yielding a fault-coverage $(\text{f-c})_J$. Of the remaining set of vectors, V_{rem} = {all possible vectors – $(V)_J$} on the hypercube H_m, choose a vector V_{next} at random and check if,

$$D\{(V)_J, V_{next}\} \geq \text{Threshold distance} \tag{10}$$

(The threshold distance is chosen by the user). If yes, then V_{next} is a candidate for V_{new}. If no, then choose from V_{rem} another vector at random and check if eqn. (10) is satisfied. Repeat till a candidate V_{new} is found. This is basically a greedy search technique where from the set V_{rem}, the first vector encountered which is farther from $(V)_J$ by a critical value is chosen as V_{new}. On the other hand, systematically searching through all the vectors in $V_{rem} = 2^m - (V)_J$ (for $m > 50$ and $(V)_J \approx 100$ to 10,000) to find the vector farthest from $(V)_J$ would be very time consuming and hence the greedy search technique is resorted to.

The pseudo-code for the complete TVG algorithm is as follows:

$(V)_J \leftarrow$ Initial seed vectors

2. Estimate $(\text{f-c})_J$
3. While $(\text{f-c})_J <$ (Required value)
   begin
   Find V_J^{cm} from $(V)_J$
3.1 Generate a vector V_{next} from V_{rem} randomly
   If $\{d(V)_J^{cm}, V_{next}\} >$ Threshold)
3.2 begin
   Remove V_{next} from V_{rem}
   end
   Estimate $(\text{f-c})_{new}$
3.3 If $\{(\text{f-c})_{new} > (\text{f-c})_J\}$
   begin
   $(V)_J \leftarrow (V)_J + (V_{new})$
   $(\text{f-c})_J \leftarrow (\text{f-c})_{new}$
   end
   else
   go to step 3.1.
   end
4 $(V)_J$ is the required set of TVs.

4.2 Method B

A more deterministic method can be set up to find the vector farthest from a set of vectors $(V)_J$.

(i) Compute V_J^{cm} from $(V)_J$ as given in Eqns. (5) and (6). Obviously each coordinate C_i of V_J^{cm} as defined in Eqn (6) is a fraction between 0 and 1.

(ii) Convert V_J^{cm} into a binary vector V_J^{bin} as follows.

$$V_J^{bin} = (b_1, b_2, \ldots\ldots b_m)$$

where $0 \leq c_i < 0.5 \Rightarrow b_i = 0$,
$c_i = 0.5 \Rightarrow b_i = 0$ or 1,
with equal probability
$0.5 < c_i \leq 1.0 \Rightarrow b_i = 1$
for $i = 1 \ldots\ldots m$ (11)

Thus, we generate a binary vector V_J^{bin} from V_J^{cm}, where the components of the m-tuple V_J^{bin} will be either 0 or 1. Hence V_J^{bin} will be one of the vertices of H_m. It physically denotes that vertex of H_m which is closest to the centre-of-mass of $(V)_J$.

(iii) It is then obvious that, since V_J^{bin} is a representative of the set $(V)_J$, the complement of V_J^{bin}, denoted by $\bar{V}_J^{bin}$, will denote the vector which is farthest from V_J^{cm} i.e. $D\{(V)_J, \bar{V}_J^{bin}\}$ is maximum.

(iv) If V_J^{bin} happens to belong to $(V)_J$, then suitably generate vectors which are unit Hamming distance from $\bar{V}_J^{bin}$. Each one of these is a candidate for V_{next}.

v) In the event of all the neighbours of $\bar{V}_J^{bin}$ belonging to the already generated set $(V)_J$, then V_{next} is generated randomly as suggested in A. Having chosen V_{new}, check if it increases $(\text{f-c})_J$. If yes, include V_{new} in $(V)_J$. If no, then choose another V_{new}. This method does not involve: (a) randomly generating vectors from $V_{rem} = 2^m - (V)_J$, and (b) checking for the threshold value condition, as suggested in A.

The pseudo-code of *B* is given in the following :

1. $(V)_J \leftarrow$ Initial seed vectors.
2. Estimate $(\text{f-c})_J$
3. While $((\text{f-c})_J <$ Required value)
   begin
3.1 Compute centre of mass (cm) in binary form V_J^{bin}
3.2 Find complement of cm $\bar{V}_J^{bin}$ as next probable vector
3.2.0 If (the next probable vector $(V)_J$
   begin
   Compute $(\text{f-c})_{new}$
3.2.1 If $\{(\text{f-c})_{new} > (\text{f-c})_J\}$
   begin
   $(V)_J \leftarrow (V)_J + \bar{V}_J^{bin}$
   $(\text{f-c})_J \leftarrow (\text{f-c})_{new}$
   go to 3.1
   end
   else
   go to 3.2.2
   end
   else
   begin
3.2.2 If all 1-neighbours not exhausted
   begin
   Generate a 1-neighbour
   go to step 3.2.0
   end
   else
   Randomly generate next vector
   end
4. $(V)_J$ is the required set of TVs.

5. IMPLEMENTATION AND PERFORMANCE RESULTS

Some preliminary investigations of A and B described above were carried out to generate TV's for different circuits having different functionalities. These circuits for (A) are : some of the blocks of processors designed at ANURAG (where many VLSI chips required by the various DRDO laboratories are being designed), and for (B): the ISCAS'85 benchmark circuits, and may be taken as representatives of their class.

Brief descriptions of the circuits follow:

5.1 ANURAG Designed Circuits

(a) Opcode decoder

Decodes the opcode in an assembly instruction. The input is a 13-bit instruction:

inst (31:25),
inst (23:22).

The output are signals showing the type of ruction. The ircuit is shown in Fig. 1.

(b) Adder-subtractor

This is a 32-bit block performing addition and subtraction for both signed and unsigned cases. The inputs to the block are:

a(31:0)	(operand 1)
b(31:0)	(operand 2)
cin	(carry-in)
WS	(0=unsigned, 1=signed)
WC	(0=without carry, 1=with carry)
asm	(0=addition, 1=subtraction).

he outputs are

s(31:0)	(sum)
	(carry-out)
	(overflow flag)
	(negative flag)
	(zero flag).

he c it is shown in ig

(c) Logical unit

This block performs all logical operations (eg. AND, OR,). The core implements the following Boolean expression:

Result(1-bit) = XOR(O3,NAND NAND(Y,X
NAND [NAND(,!04),!X,O2]

where
X,Y are inputs, O_i's are control sig als specifying the operation to be performed.

This core is repeated 32 times for 32-bit logic operation.

The O_i's are

Operation	O4	O3	O2	O1
AND	0	0	0	
OR	0	1		
XOR	0	1		
NOT(X)	1	0		
NOR	0	0		0
XNOR	0	0		
NAND	0	1	0	

The circuit is shown in Fig. 3.

(d) Sign-extension logic

This block extends sign bit from the specified bit positions: (7, 9, 11, 13, 15, 23).

The inputs are:

D(31:0)	(32-bit data)
B7, B9, ... B23	(sign bit position)

The output is

D0(0:31	(32-bit sign-extended data

The circuit is shown in

To assess the performances of A and B, they were compared with TV's generated by the random method (R). In R, each input vector V_{new} is generated randomly and completely independent of the previous vectors generated so far. It represents picking any point on the hypercube H_m with equal probability. It is desirable that A and B yield better performances than R. The results of the analysis on the various circuits using R, A and B are presented in Table 1.

In Table 1, we give the circuit name, number of testable faults, which can be detected by fault-simulator, the number of vectors generated or tried (the set V_{new}), the number of vectors selected out of V_{new} which increase the f-c (the f-c after saturation) i.e., when it plateaued out. These figures are presented for A, B and R.

It can be seen from the able that B is superior to A and R in different wa

(a) For all the circuits, the f-c obtained from B is greater than those obtained from A and R (For the sign-extension logic, the f-c from B was equal to the f-c from R and greater than f-c from A).

Table 1 Performance figures of different methods on various circuits

Circuit description	No. of detectable faults	Method	No. of vectors tested	No. of vectors selected	Fault coverage (%)
Opcode decoder	530	Random	37	17	56.59
		A	26	15	62.08
		B	71	25	75.66
Adder-subtractor	2582	Random	66	41	70.06
		A	64	14	74.48
		B	23	22	76.09
Logical-unit	1360	Random	67	33	81.47
		A	23	17	77.43
		B	104	37	92.08
Sign-extension logic	1350	Random	43	20	69.93
		A	26	11	61.18
		B	40	17	69.93

(b) In the case of logical unit, though the vectors finally selected were almost the same for R and B, the f-c was much higher for B. Hence B was able to select a better set of vectors.

(c) For the adder-subtractor, not only was the f-c for B the highest, the number of vectors generated was the least (almost one-third of those generated by A and R). Hence B utilised one-third the system time.

(d) For the logical unit, the f-c obtained from B was greater than 90 per cent.

Hence it can be concluded that B is better than both A and R.

It can also be seen from Table 1 that A and R can be compared as follows:

(i) The f-c is higher from A in half the cases.

(ii) The number of vectors finally selected by A is less than that selected by R. Hence the selected vectors are more optimally chosen by A.

(iii) The number of vectors generated by A is always less than for R. Hence A always utilises less system time.

Thus A could be said to be slightly better than R.

5.2 ISCAS'85 Benchmark Circuits

The ISCAS'85 circuits are the now well established benchmarks for combinational circuits. Method B was run on the benchmark circuits and compared with the results obtained by R on the same circuits, as reported by Kawai, *et al*[6].

It can be seen that

(a) For C 432 C 5350 and C 7550, B yields higher f-c than R for the similar number of TVs.

(b) In other cases, the f-c obtained by B was higher than R.

6. DISCUSSION AND CONCLUSION

A new algorithm has been described for TVG, based on the concept of a distance of a point on a hypercube from a given set of points on the hypercube, along with the idea of maximising the distance. Two alternative methods A and B for maximising the above distance were suggested and their performances on different circuits were compared with the random TVG method R. It was observed that B is superior to the other two methods in a variety of ways. Also, A is slightly better than R.

In B, scope for exploring neighbouring vectors unit Hamming distance from $\bar{V}_j^{bin}$ was incorporated. In principle, this can be extended to neighbours with larger Hamming distances, depending on the need. It is important to point out that for all the circuits analysed in Table 1, change by unit Hamming distance was sufficient. That is, all the neighbours unit Hamming distance from $\bar{V}_j^{bin}$ were never exhausted. One of the above neighbours always succeeded in increasing the fault-coverage.

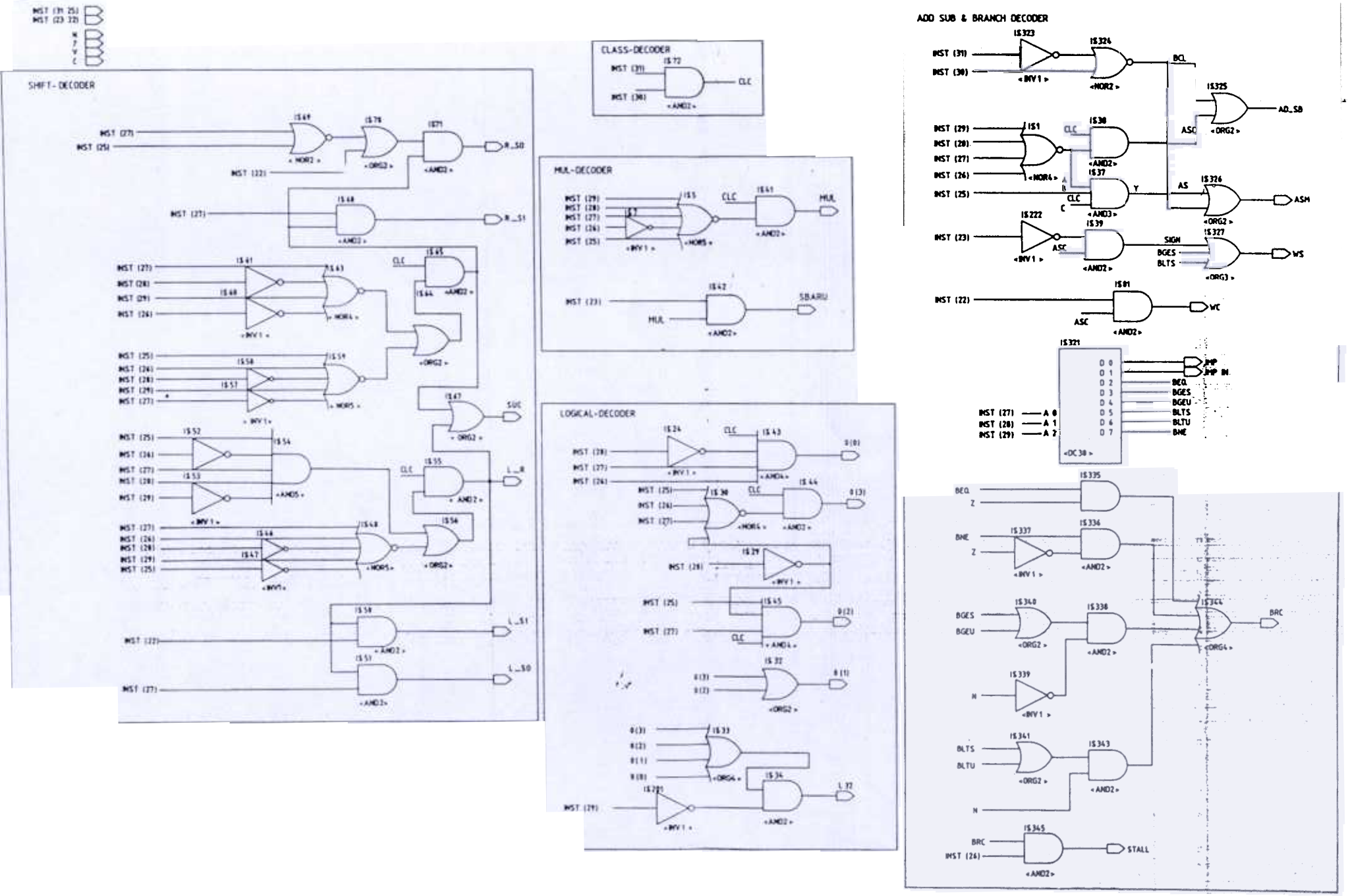


Figure 1. Opcode decoder block.

32 - Bit ADDER SUBTRACTOR

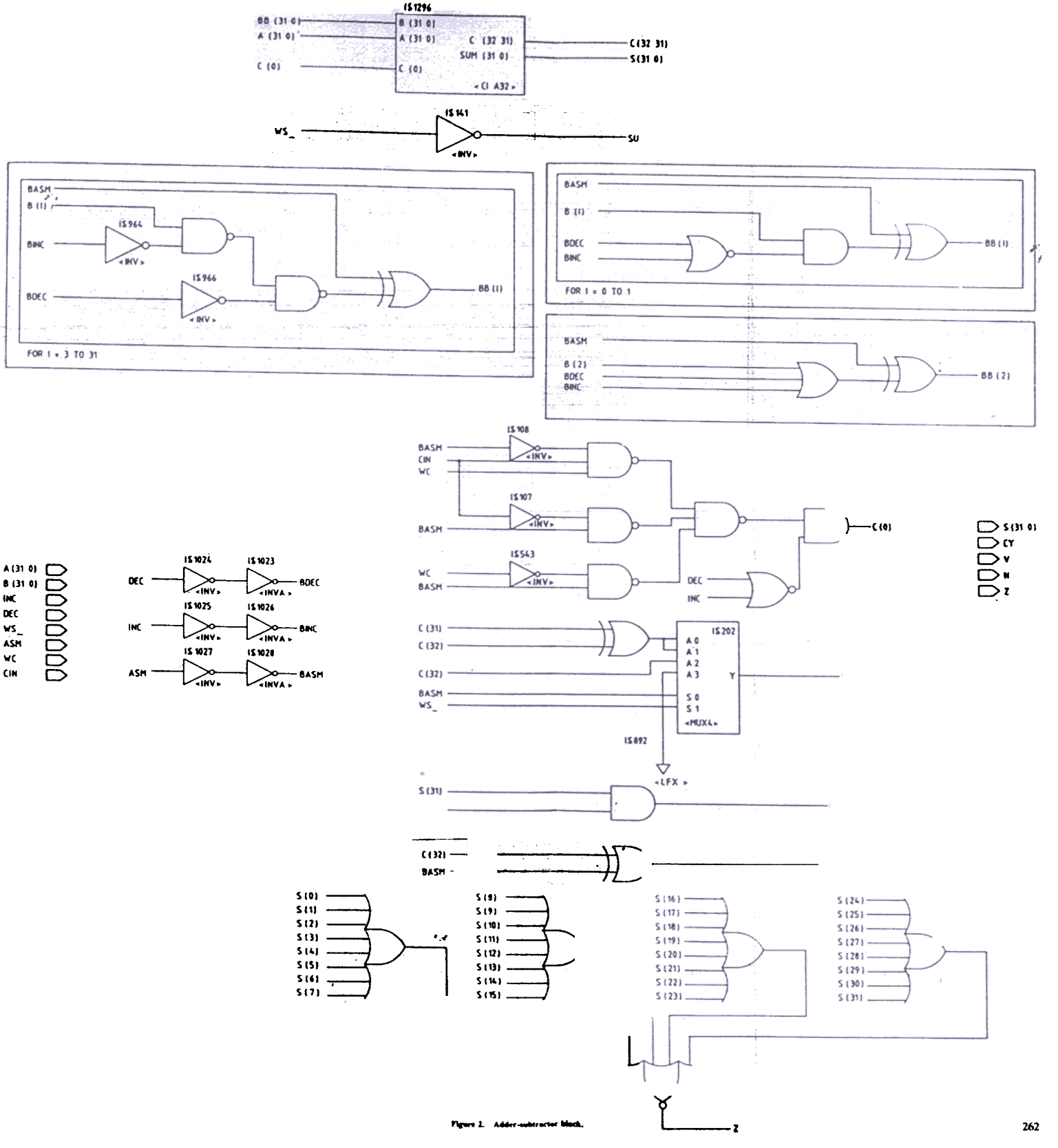
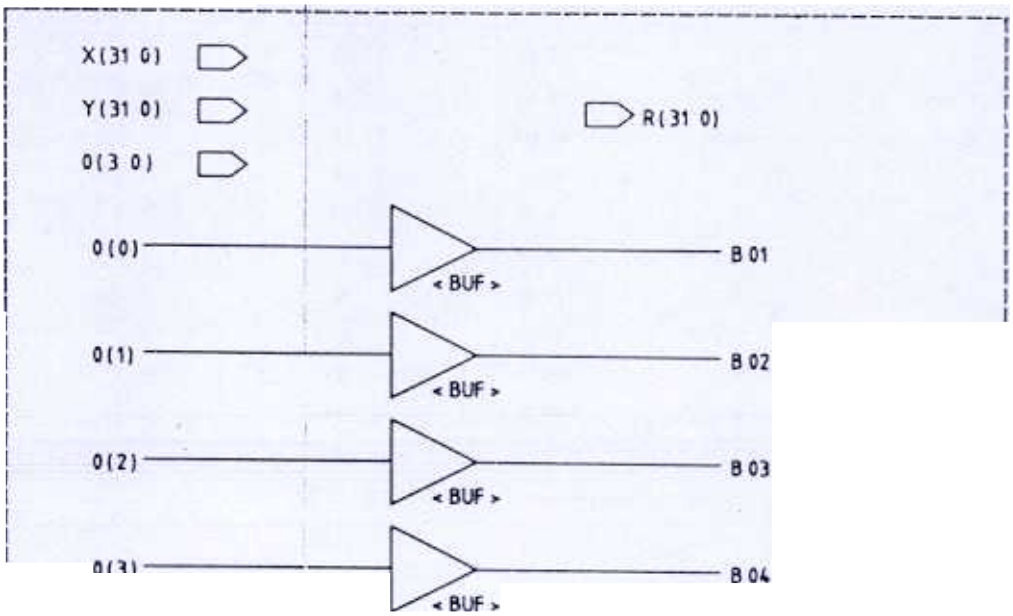
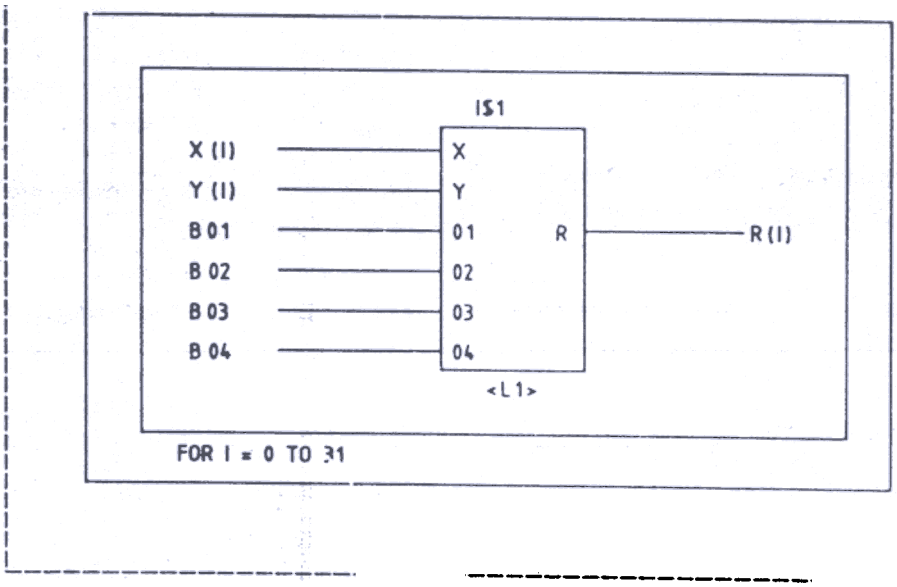


Figure 2. Adder-subtractor block.

32 - Bit LOGICAL UNIT





OPERATION	O (3 0)
AND	
NOR	
XNOR	
NAND	5
OR	
XOR	
NOT X	A

Figure 3. Logical-unit block

Table 2. Performance figures of different methods ISCAS'85 benchmark circuits

Circuit	Method B		Random Technique	
	No of selected vectors	Fault coverage (%)	No of selected vectors	Fault coverage (%)
C432	41	88.17	40	81.1
C499	50	95.65	6	63.7
C880	41	87.00	22	77.1
C1355	44	85.32	6	49.0
C1908	37	70.20	17	55.0
C2670	48	74.37	32	70.20
C3540	69	77.95	42	72.6
C5350	36	81.78	40	78.8
C6288	36	99.15	23	98.4
C7550	50	83.70	54	82.4

In the case of the opcode decoder, the total number of valid input instructions is usually small. Hence all the inputs can be exhaustively enumerated in the test-vector set to get maximal fault-coverage. The statistics shown in Table 1 and Table 2 highlight the superiority of B over the R.

It is next proposed to integrate the TVG method described in this paper with some local methods of TVG (e.g. PODEM-based algorithms). This is towards setting up a TVG package yielding fault-coverages greater than 95 per cent.

ACKNOWLEDGEMENTS

Dr K Neelakantan, Director, ANURAG, has been a source of constant encouragement and support throughout this study. The authors are very grateful to him for the same. They also thank R Bidnur and AVSS Prasad for giving the circuits. MVA is thankful to his colleague PP Ghosh for introducing him to the field of TVG and to MK Srinivas (IISc) for very useful discussions.

REFERENCES

1. Agrawal, V.D. & Seth, S.C. Test generation for VLSI chips-tutorial. IEEE Catalog No. EH0277-4, 1988.
2. Russell, G. & Sayers, I.L. Advanced simulation and test methodologies for VLSI design. Van Nostrand, London, 1989.
3. Miczo, A. Digital logic testing and simulation. John Wiley, Singapore, 1987.
4. Abraham, J.A. & Fuchs, W.K. Fault and error models for VLSI. *Proc. IEEE*, 1986, 74, 639.
5. Schurtz, D.R. & Metze, M. A new representation of faults in combinational digital circuits. *IEEE Trans. Comput., 1972*, **21-C**, 858.
6. Kawai, M.K.; Oozeki, K.; Takahashi, M.; Ono, M.; Ishizaka, Y. & Masui, T. Automatic test pattern generator for large combinational circuits. *Proc. ISCAS'85*, June 1985, pp. 663-70.

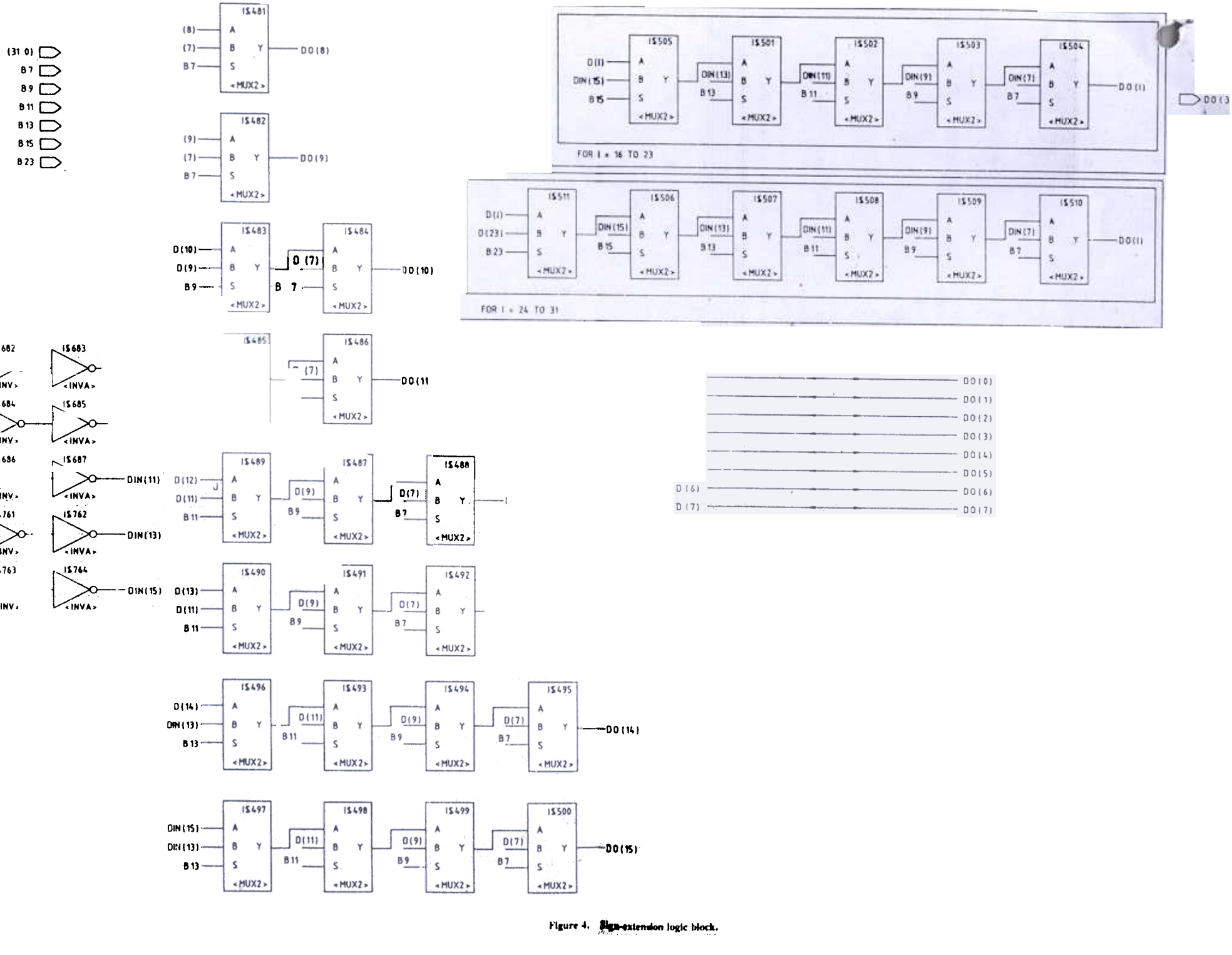


Figure 4. Sign-extension logic block.