Dynamic Fault Diagnosis on Reconfigurable Hardware †

Fatih Kocan and Daniel G. Saab
Electrical Engineering and Computer Science Department
Case Western Reserve University
Cleveland, Ohio, 44106

Abstract: In this paper, we introduce a new approach for locating and diagnosing faults in combinational circuits. The approach is based on automatically designing a circuit which implements a closest-match fault location algorithm specialized for the combinational circuit under diagnosis (CUD). This approach eliminates the need for large storage required by a software based fault diagnosis. In this paper, we show the approach's feasibility in terms of hardware resources, speed, and how it compares with software based techniques.

1 Introduction

Fault location (Diagnosis) techniques are classified into two main groups based on the analysis performed; (1) cause-effect analysis [2]; (2) effect-cause analysis [7,10]. In those techniques, the accuracy level of locating a fault for a given test set is measured by the Diagnostic Resolution (DR) which is the fraction of distinguished fault pairs resulting from the application of the test set. The measure DR is controlled by the size/number of partitionned faults (Faults in a partition are pair-wise indistinguishable) and the quality of the test set[6].

In cause-effect analysis [4,9], the logical behavior of primary outputs in the presence of faults are computed using fault simulation and stored in a table referred to as fault dictionary [9]. Faults are located by comparing the stored behavior to the response of the CUD. The location of the fault that closely matches the behavior of the CUD is reported as the suspect location. It is possible that more than one location can be a suspect. In this case, suspect locations are ranked by their 'closeness' to the response of CUD.

† This work was supported by NSF (Contract: MIP-9528931) and by Gebze Institute of Technology, Turkey.

DAC 99, New Orleans, Louisiana

Fault dictionaries are classified according to the type of information they store. On one hand, A pass/fail dictionary stores for each vector and fault a pass/fail entry. A pass/fail entry means that the logic values of primary outputs in the fault-free circuit are equal/different to/from those of the faulty-circuit. On the other hand, a full fault dictionary stores for each vector the logic values of all outputs for each fault. In any case, the sizes of these dictionaries depend on the number of faults, number of outputs, and the size of test set. In general, fault dictionaries require large storage. To alleviate memory requirement compaction and compression techniques are used to reduce dictionary size [3,4,5]. Storage requirement can also be reduced by re-computing the information stored in the dictionary when needed. Techniques that use this method are referred to as dynamic fault diagnosis [3]. The computation cost for dynamic diagnosis is high which limit its usage to the case where only small number of diagnosis is performed [3]. To further reduce the overhead, a technique which mixes dynamic and dictionary based diagnosis is proposed in [4].

In Effect-cause analysis, faults are located by observing the faulty behavior of the CUD and deducing the location of the defect from the observed faulty values and the fault-free behavior. This process involves implications of logic values in both forward and backward and vertically across vectors [10,16].

Emulation systems are being used increasingly in the design, verification, and in rapid proto-typing of digital systems [12]. To increase the use of these emulation systems, several methods propose ways to emulate computer-aided design algorithms such as serial fault simulation [13], critical path tracing [19], PODEM [1], and Satisfiability [1,20,18].

In this paper we introduce an approach for locating/diagnosing faults in combinational circuits that uses emulation systems. This approach eliminates the need for large storage, reduces the CPU time required by a software-based diagnosis, and preserve diagnostic resolution of a full dictionary. This paper is organized as follow: Section 2 describes the dynamic diagnosis procedure. In Section 3, we present implementation details. In Section 4, results are presented. Discussions and conclusions are presented in Section 5.

2 Dynamic Fault Location

In dynamic fault location, fault simulation is used repeatedly to produce the faulty circuit response when needed. To locate the cause of the failure, the logical behavior of the faulty circuit is compared to the logical behavior of the CUD and one of following two algorithms is applied to the result. The first algorithm is based on the best-matched behavior and the second is based on fault dropping. For the best match algorithm [4], the number of matches between the faulty circuit and the CUD is kept for each fault. Faults are then sorted in descending order where the first ranked is the fault that resulted in the most number of matches. This ranking indicates suspect fault location (first being best). The second technique is based on eliminating any fault from being a suspect if for any vector in the test set the behavior of the outputs of the faulty circuit does not match exactly the behavior of the output of the CUD. This technique is very fast and relies on the premise that the logical behavior of every failure is equivalent to at least one fault[3] for the given test set. In general, this is not the case.

In our implementation, we use a combination of the two techniques which is shown in Figure 1. This algorithm computes for each fault the behavior of the faulty circuit and the CUD. The primary outputs are compared to compute the number of matches (Match variable) and to keep track of the number of vectors with partially matched behavior in (Failed_vector variable). The variable Failed_vector is incremented whenever the logical behavior of the faulty circuit partially matches that of the CUD. In the case when the faulty behavior matches exactly that of the circuit under test for all test vectors, the variable Failed_vector remains zero. In the other case, the variable Failed_vector is used to prioritize faults with equal number matches (Match). The routine Insert_Sort updates the sorted list of suspect faults L. The dynamic fault location algorithm is shown in Figure 1. In the implementation of the algorithm, the computation of the logic behavior of the CUD, the emulation of a faulty circuit, and the computation of the number of matches are all performed in one clock cycle. Incrementing variable Failed_vector may requires an extra clock cycle. This means that the inner loop needs at most two clock cycles. The insertion procedure (Insert_Sort) requires more than two clock cycles. This is discussed in the next section.

Figure 2 shows the detail of Procedure Insert_Sort. This procedure receives a list L and a fault with corresponding Match and Failed_vector. This procedure inserts the fault in descending order in list L.

3 Implementation

Figure 3 shows a high-level diagram for the overall implementation. This model consists of three main components: a computation block which implements the fault

```
Procedure Dynamic_Diagnosis
Input:  C_f Faulty circuit model.
        T The set of diagnosis test vectors.
        F The set of all modeled faults.
        R_f(t) Faulty response for test t
        R_cud(t) Fault-free response for test t
Output :  L Sorted list of suspect locations.

    L = empty;
    for each fault f in F do {
        Match = 0 ;
        Failed_vector = 0 ;
        C_f  =  Inject(f,C) ;
        For each test t in T do {
            R_cud(t)  = Apply(CUD) ;
            R_f(t)  = Emulate(C_f ) ;
            D = number-of-ones in((R_cud(t) ⊙ R_f(t)));
            Match = Match + D;
            if (D != |R|) Failed_vector++;
        }
        Insert_Sort(L, <f,Match,Failed_vector>);
    }
```

Figure 1: Dynamic Fault Diagnosis Procedure

simulation and matching, an insertion block which maintains a sorted list of suspect locations and a controller that manages the flow of the computation. This implementation relies on signals, which are generated in one of these blocks and managed by the controller. In addition, the controller loads the number of test vectors into a register during initialization and it starts the execution of the algorithm by activating the 'Start' signal of the computation block. When the computation block receives this signal, it computes Match, Fault_ID, and Failed_Vector values and sends a 'Done' signal to the controller. Consequently, The controller activates the insertion-block 'Start' signal and waits for the insertion-block to update the list of suspect faults. The insertion-block activates its 'Done' signal when this process is completed. The controller repeats this process for every fault. Upon completion, the list of suspect faults is traversed sequentially and printed.

[ht]

The details of computation block are depicted in Figure 4. This block implements the fault simulation and computes the number of matches and the number of faild vectors. This block consists of a 'FAULT-INJECTION CIRCUIT', a 'CIRCUIT UNDER DIAGNOSIS', a 'MATCHER', a Partial match tester, a TESTVECTORS RAM and three registers. The 'FAULT INJECTION CIRCUIT' is similar to the one proposed in [13]. This circuit consists of a fault activation and a fault injection circuit as shown in Figure 5. The fault activator is a shift register where each Flip-Flop (FF) in this register corresponds to a fault. In this representation, when a FF is set to logic '1' (high) the corresponding faults

```
Procedure Insert_Sort
Input :  L, <f,Match,Failed_vector>
Output :  Updated L

    Curr = Head(L); Pev = Curr; found = FALSE;
    while (Curr != 0 and found == FALSE) do {
        if(Match<Curr.Match) or (Match==Curr.Match
            and Failed_vector>Curr.Failed_Vector){
            Prev = Curr; Curr = Curr->Next;
        } else { found = TRUE; Insert fault;}
    }
    If (found==FALSE) append fault at tail of L;
```

Figure 2: Insert_Sort Algorithm

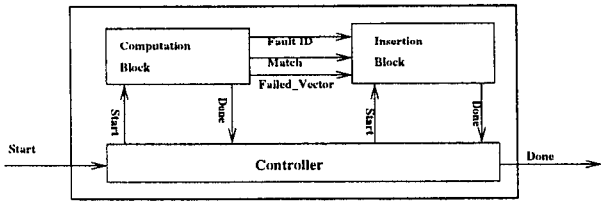

Figure 3: Block Diagram of the Hardware Model.

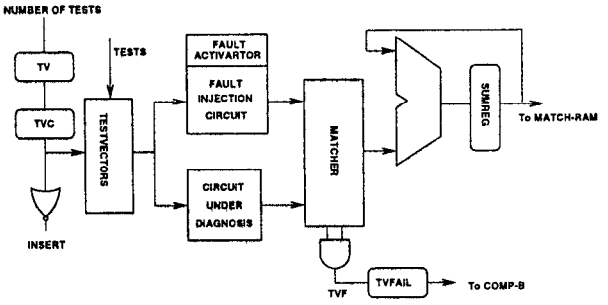

Figure 4: The computation block.

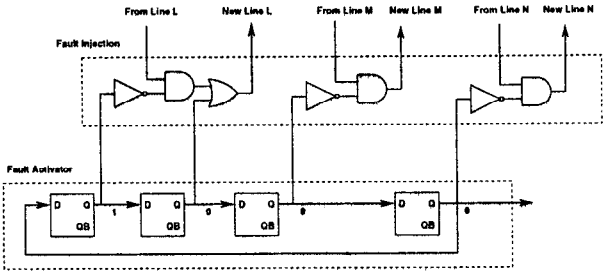

Figure 5: Fault activation and injection circuits.

will be injected into the circuit as shown in Figure 5. In this figure, the circuit for injecting four faults (L stuck-at-0 (s-a-0), L stuck-at-1 (s-a-1), M s-a-0 and N s-a-0) is shown. Each of these faults is activated by setting the corresponding FF to '1' which modifies the logic behavior of the circuit by fixing the logic value of the fault location to constant '0' ('1') for s-a-0 (s-a-1).

The fault activation circuit minimizes the number of FPGA reconfiguration needed to compute the suspect location and reduces the total run time. The fault injection and activation circuits increase the size of the emulated circuit. This increase is computed as follow: given 'p' s-a-0 faults and 'q' s-a-1 faults in an 'n' gate circuit, the size of the emulated circuit increases by 'p+q+1' FF's and '2*p + q' gates.

This response of the CUD can either be stored or emulated. In this implementation, we emulated the faulty response ('CIRCUIT UNDER DIAGNOSIS' component) by injecting failure into the circuit. The Matcher circuit computes the number matches between the primary output responses of the faulty circuit and the primary output response of the emulated CUD circuit. To do this, both circuits are subjected to the same vectors, which are stored in a RAM. For every vector, the number of matches is computed by XNORing the corresponding primary outputs of both CUD and the faulty circuit, and adding the number of '1's in the corresponding XNOR outputs. A Matcher circuit for an n primary-outputs circuit is depicted in Figure 6.

The result of the addition is accumulated into register SUMREG which at the end would contain the total number of matches for a fault. To compute the number of failed-vectors, the output of corresponding XNOR gates are logically ANDed to generate Test Vector FAILED (TVFAILED) which is used as a control to a counter (TVFAIL) which counts the number vectors with partially matched outputs. Register TV holds the number of test vectors which is loaded at the beginning of a diagnosis session. The test vectors are also loaded into the 'TESTVECTORS' RAM at the beginning. When activating a new fault, the content of TV is loaded into register TVC which indicates which test vector is being applied to the circuit. TVC is decremented after the application of every test and when it becomes '0' it activates the fault insertion circuit (Insert_Sort algorithm) by asserting signal INSERT.

The detailed design of the Insert_Sort circuit is depicted in Figure 7. The circuit consists of three RAMs, six registers, three comparators and six multiplexors. The FAILED-VECTORS RAM holds the number of partially matched vectors for each fault. The MATCH-COUNTS RAM holds the number of matches for each fault. The data-in lines of the MATCH-COUNTS RAM are connected to the register SUMREG of the Matcher circuit described in the previous section. The LINKS RAM is addressed by a fault ID (current fault) and each address contains a fault ID (next fault) that corresponds to the fault with number of matches smaller or equal to that of the current fault. The registers ID, HEAD, CURRENT, and PREVIOUS are used as pointers to facilitate the in-

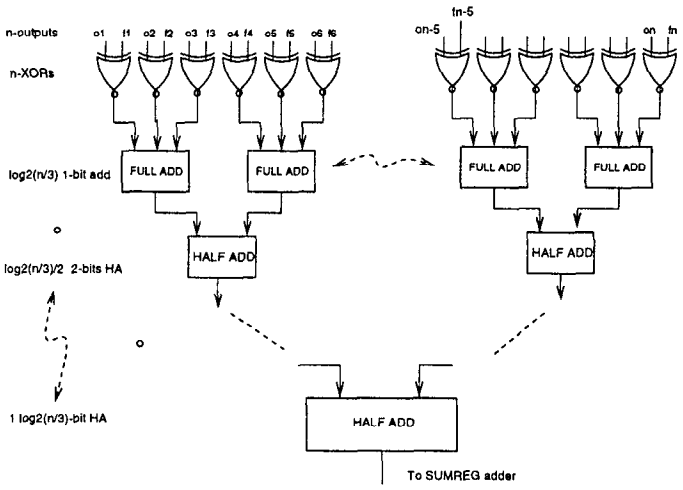

Figure 6: A Matcher circuit for an n-outputs CUD circuit.

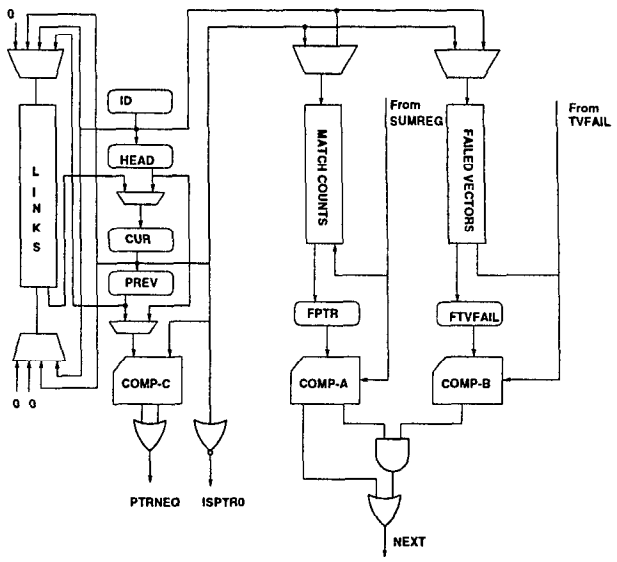

Figure 7: The Insertion block.

sertion process. The current fault identification is stored in register ID. The address of the head of the sorted list is stored in register HEAD. To traverse the list, the content of HEAD register is loaded into register CURRENT and PREVIOUS and when updating PREVIOUS and CURRENT will be pointing to two consecutive elements of the list. These are used for insertion. An element of the list consists of the content of two words one from the MATCH-COUNT and the other from FAILED VECTORS. Both of these words are addressed by register CURRENT. When traversing the list, the status signals, namely PTRNEQ, ISPTR0 and NEXT, are generated. PTRNEQ is used to test pointer equality of either CURRENT and HEAD or CURRENT and PREVIOUS. The status signal ISPTR0 is asserted when traversing reaches to the end of the list. The NEXT signal is asserted when the new fault addressed by ID has lower matches than the matches of the fault in the list pointed by CURRENT. Here, in case matches are equal, the Failed_Vector values of the corresponding faults are used to decide the NEXT signal.

The hardware cost of the dynamic diagnosis circuit is given in terms of the number of gates, FFs, and size of memory. This is given in term of the sizes of three blocks as in Figure 3: The controller, the computation, and the matcher. The controller requires 19 states regardless of CUD size. The cost of the datapath (the computation and the insertion blocks) depends on the CUD size. This can be estimated in terms of the number of test vectors (T), the number of faults (F), the number of primary inputs (PI) and the number of primary outputs (PO) of a particular CUD. Below we give the size of various memories and the number of FFs in various registers.

RAMs sizes in (wordsize in bits)x(address range):

TEST-VECTORS size is PI x (0 to log_2(T))
FAILED-VECTORS size is log_2(T) x (0 to log_2(F))
LINKS size is log_2(F)x(0 to log_2(F))
MATCH-COUNTS size is log_2(F) x (0 to log_2(T*PO))

The number of FFs:

TV, TVC each consists of log_2(T) Flip-Flops (FF).
SUMREG, FPTR each consists of log_2(T*PO) FF.
TVFAIL, FTVFAIL each consists of log_2(T) FF.
FAULT-ACTIVATOR consists of p+q FF.
ID, HEAD, CUR, PREV each consists of log_2(F) FF.

The sizes of the multiplexors and comparators can be deduced from their connections. For example, CAMP_A comparator has a size of log_2(T*PO) bits like that of FPTR. The number of connections and the number of select lines can be used to estimate the sizes of the multiplexors used.

To estimate the maximum number of clock cycles needed to diagnose a circuit, two assumption are used; (1) all test vectors are partially matched and (2) faults are always inserted at the end of the list. These assumptions result in the maximum number of cycles. Under the above assumptions, the corresponding paths and loops of the state diagram of the controller are analyzed. The computation block and the insertion block require maximum F(2T+1) and F(F+3) clock cycles, respectively. So the maximum number of clock cycles for dynamic diagnosis is F(2T+F+4) and can be written as O(2TF+F^2+4F).

To estimate the average number of clock cycles, the two earlier assumptions are modified slightly. We can assume that only half of the test vectors fails for all faults and instead of appending at the end of the list, the faults are assumed to be inserted in the middle of the list. Under these modified assumptions, the computation block and the insertion block require F(3/4T+1) and 3/4F^2+1/2F average clock cycles respectively. The average clock cycles

can be written as (3/4FT+3/4F^2+3/2F). The F^2 terms in these formulae make the number of faults a dominant factor in determining the diagnosis times. Moreover, with the help of the underlying emulation technique, the dependency on the number of primary outputs is eliminated from the computation time. In contrast, software based computation time depends also the number of primary outputs in addition to the number of test vectors and the number of faults. The software matching algorithm requires fault simulation without dropping to create the dictionary, O(F+2*T*PO*F) CPU time to compute the matches, and an average (F*log(F)) CPU time to sort the results to compute the ranking.

4 Results

We are interested in finding the hardware and computation time requirements for the proposed dynamic diagnosis technique. Since these requirements vary from one circuit to another, we use the Iscas-85 benchmark circuits to estimate the hardware requirements in term of the number of 4-input CLBs.

In addition to preprocessing times which include circuit generation and technology mapping, the worst case and average case computation times targeting an FPGA operating at 1MHz to dynamically locate faults with and without best matching feature (with/without Insert_Sort procedure) are shown in Table 1. These results are for locally collapsed faults. For example, c432 requires 0.3sec of CPU time to generate the diagnosis circuit, 25.4 seconds of CPU time to map into 4-input CLBs, (0.226 seconds with insert)/(0.020 seconds without insert) on the average to run the mapped diagnosis circuit on an FPGA running at 1MHz, (0.329 seconds with insert)/(0.052 seconds without insert) maximum to run the mapped diagnosis circuit on an FPGA running 1MHz.

Table 1: Time required by the dynamic diagnosis circuits

Ckt	Prep Gen/Map (ms)/(sec)	Avg (sec) w Insert	Avg (sec) w/o Insert	Max (sec) w Insert	Max (sec) w/o Insert
C432	0.3/25.4	0.226	0.020	0.329	0.052
C499	0.4/34.2	0.461	0.030	0.656	0.079
C880	0.6/41.6	0.688	0.022	0.949	0.059
C1355	1.0/38.9	2.011	0.152	2.886	0.404
C1908	1.1/60.0	2.866	0.217	4.113	0.576
C2670	2.3/68.0	5.999	0.338	8.452	0.898
C3540	3.2/77.9	9.361	0.545	13.211	1.450
C5315	7.0/136.1	22.233	0.763	30.660	2.027
C6288	13.4/292.7	45.209	0.228	60.589	0.596
C7552	12.7/210.3	44.501	1.745	61.668	4.643

To find the average time, the formulae found in Section 3 are used. These formulae are directly extracted from state diagram of the design. When all faults are modeled and the Insert_Sort is not activated, the computation block is the sole determining factor for the computation time. The columns (w. Insert) and (w/o. Insert) correspond to the design with best matching feature and without best matching feature respectively. From that table, a fault can be located dynamically in at most 45 seconds on the average and in 1 minutes for the worst case when best matching is utilized (not considering the preprocessing time). In the cases where all possible faults are modeled, faults can be located at most in 1.75 seconds on the average and 4.6 seconds for the worst case. The numbers under (w/o Insert columns) are the required time to create a full dictionary for the corresponding benchmarks. In this approach, The Insert_Sort consumes more than 83 and 92 percent of total computation time in average case and worst case respectively.

Table 2. shows the run time for software based diagnosis. The time shown under Fault-simulation includes both the CPU time for fault-simulation and the dictionary creation time. The time shown under Match is The CPU time for generating the matching degree/ranks. The last column reports the hardware speedup. Note, that for most circuits the speedup is in the order of 1000x.

Table 2: Software Run Time and Speedup

Circuits	Fault Simulation (sec)	Match	Speedup Software/ FPGA
C432	23.9	1.410	1255.576
C499	37.3	9.49	1544.525
C880	21.2	5.750	1180.179
C1355	93.2	48.430	927.966
C1908	157.7	54.340	975.2453
C2670	382.2	471.680	2522.359
C3540	747.8	120.190	1590.21
C5315	547.7	939.060	1947.014
C6288	443.2	70.910	2251.541
C7552	1924.5	1890.020	2184.969

Hardware requirements for the design with and without the Circuit under Diagnosis (see Figure 4) component are computed. The SIS technology mapping tool [17] is used to compute the number of 4-input CLBs required and the time to map the circuit into FPGAs. The numbers of CLBs are obtained by excluding the storage cost of the RAMs used (see Figure 4 and Figure 7). The results are tabulated in Table 3.

The average CUD hardware usage is 9.11 percent of the overall hardware usage for the benchmark circuits. Besides, if the CLB requirement is too high due to the size of fault injected circuit when the number of faults to be injected is high, then multi-configuration can be used to inject a small set of faults per configuration. When multi-configuration is needed, only the faulty-circuit (see

Table 3: CLBs required for mapping the circuits

Circuit	Num of CLBs w. CUD	Num of CLBs. w/o. CUD
C432	3580	3353
C499	4515	4277
C880	5099	4689
C1355	6828	6278
C1908	8151	7280
C2670	11949	10869
C3540	13527	11908
C5315	20174	17948
C6288	28290	25874
C7552	27101	23811

Figure 4) can be reconfigured and the other components in the computation block, the controller and the insertion block may not be reconfigured.

5 Conclusions and Future Work

In this paper we introduced a new emulation approach for locating and diagnosing faults in combinational circuits. The approach is based on automatically designing a circuit which implements a closest-match fault location algorithm specialized for the CUD. We showed the feasibility of this approach in terms of hardware resources, speed, and compared with software techniques. We showed an improvements in term of speeds over software based fault location.

References

[1] M. Abramovici and D. G. Saab, "Satisfiability on Reconfigurable Hardware," Sevent International Workshop on Field Programmable Logic and Applications, September 1997.

[2] M. Abramovici, M. A. Breuer and A. D. Friedman, Digital Systems Testing and Testable Design , IEEE Press, 1994.

[3] I. Pomeranz and S. M. Reddy, "On Dictionary-Based Fault Location in Digital Logic Circuits," IEEE Trans. on Computers, vol. 46, no. 1, pp. 48-59, January 1997.

[4] P. G. Ryan, Shishpal Rawat and W. K. Fuchs, "Two-Stage Fault Location," Proc. 1991 ITC, pp. 963-968.

[5] P. G. Ryan, W. K. Fuchs and I. Pomeranz, "Fault Dictionary Compression and Equivalence Class Computation for Sequential Circuits," Proc. 1993 ICCAD, pp. 508-511.

[6] K. Kubiak, S. Parkes, W. K. Fuchs and R. Saleh, "Exact Evaluation of Diagnostic Test Resolution," Proc. 1992 DAC, pp. 347-352.

[7] M. Abromovici, "A Maximal Resolution Guided-Probe Testing Algorithm," Proc. 1981 DAC, pp. 189-195.

[8] I. Hartanto, V. Boppana and W. K. Fuchs, "Diagnostic Fault Equivalence Identification Using Redundancy Information & Structural Analysis," Proc. 1996 ITC, pp. 294-301.

[9] J. Richman and K. R. Bowden, "The Modern Fault Dictionary," Proc. 1985 ITC, pp. 696-702.

[10] H. Cox and J. Rajski, "A Method of Fault Analysis for Test Generation and Fault Diagnosis," IEEE Trans. CAD, pp. 813-833, July 1988.

[11] L. Burgun, F. Reblewski, G. Fenelon, J. Barbier, and O. Lepape, " Serail Fault Simulation," Proc. 1996 DAC, pp. 801-806.

[12] M. Butts, J. Bacheler, and J. Varghese, "An Efficient Logic Emulation System," Proc. 1992 ICCD, pp. 138-141.

[13] K.-T Cheng, S.-Y Huang, and W.-J Dai, "Fault Emulation: A New Approach to Fault Grading," Proc. 1995 ICCAD, pp. 681-686, Nov.

[14] I. Pomeranz, and S.M. Reddy, "On The Generation of Small Dictionaries for Fault Location," Proc. 1992 ICCAD, pp. 272-279, Nov.

[15] K. Takayama, F. Hirose, and N. Kawato, "A Test Generation System Using a Logic Simulation Engine," FUJITSU Sci. Tech. J., 27, 3, pp. 285-289, September 1991.

[16] M. Abramovici, and M. A. Breuer, "Fault Diagnosis Based on Effect-Cause Analysis,", Proc. 1980 DAC, pp. 69-76.

[17] R. Murgai, N. Shenoy, R. K. Brayton, and A. Sagiovanni Vincetelli., "Improved Logic Synthesis Algorithms for Table Look Up Architecture," Proc. 1991 ICCAD, pp. 564-567.

[18] P. Zhong, M. Martonosi, P. Ashar, and S. Malik "Using Reconfigurable Computing techniques to Accelerate Problem in th CAD Domain: A Case Study with Boolean Satisfiability," Proc. 1998 DAC.

[19] M. Abramovici and P. Menon, "Fault Simulation on Reconfigurable Hardware," Proc. IEEE Symp. On FCCM, April 1997.

[20] T. Suyama, M. Yokoo, and H. Sawada, "Solving Satisfiability Problems on FPGAs," Proc. Intn'l. Workshop on Field-Programmable Logic and Applications , 1996.

[21] R.G. Wood and R.A. Rutenbar, "FPGA Routing and Routability Estimation Via Boolean Satisfiability," Proc. Intn'l. Symp. On FPGAs, February 1997.