Dynamic Learning Based Scan Chain Diagnosis

Yu Huang
Mentor Graphics Corporation, 300 Nickerson Road, Marlborough, MA 01752, USA

Abstract

Scan chain defect diagnosis is important to silicon debug and yield enhancement. Traditional simulation-based chain diagnosis algorithms may take long run time if a large number of simulations are required. In this paper, a novel dynamic learning based scan chain diagnosis is proposed to speedup the diagnosis run time. Experimental results illustrate that by using the proposed dynamic learning techniques, the diagnosis run time can be reduced about 10X on average.

1. Introduction

ATPG patterns that utilize scan chains to provide stimulus and capture responses from circuitry have become the primary method for achieving test coverage in digital logic circuits. Scan based testing has proven to be a cost effective method to achieve good test coverage with minimal test time and pattern development overhead. Meanwhile, scan based diagnosis has also opened up new avenues for failure analysis. Scan diagnosis can be utilized to quickly isolate the suspects and provide efficient guidance to physical failure analysis (PFA).

The Achilles' heel for the application of scan-based diagnosis is the integrity of the scan chains. The amount of die area consumed by the scan flops, scan chain connections, and scan control circuitry can range from 15-30% [GUO01]. The number of die failing the scan chain integrity test will typically scale proportionally with the percentage of total circuitry involved with the scan chains. Several previous papers [KUN94][CRO05] did a good job of describing the challenges of diagnosing the scan chain defects. One of the very important issues of the chain diagnosis is the run time. With the requirement of volume diagnosis in the manufacture environment for yield improvement, the diagnosis run time becomes even more important.

In this paper, the run time bottleneck of the previously published chain diagnosis algorithms is identified. Several novel dynamic learning techniques are proposed. We will illustrate later in the experimental results that applying the new dynamic learning techniques can reduce the chain diagnosis run time about one order of magnitude on average.

The rest of this paper is organized as follows. Section 2 reviews prior art of chain diagnosis algorithms. In Section 3, dynamic learning based chain diagnosis is proposed to boost chain diagnosis speed. Section 4 presents experimental results and conclusions are drawn in Section 5.

2. Review of chain diagnosis

The previous chain diagnosis methodologies can be classified into two categories: hardware-based and software-based. Hardware-based methods use some special scan chain and scan cell designs to facilitate the diagnosis process [SCH92] [WU98] [EDI95] [NAR97]. In this paper, we do not consider hardware based chain diagnosis methods because normally these methods incur some hardware overhead that prevents them from being applied in practice.

Software-based algorithms do not need any modification of the basic scan circuitry [KUN94] [GUO01] [STA01] [HUA03]. In [KUN94], it generates special patterns to diagnose chain failures. In [GUO01] and [STA01] fault simulation and matching algorithms are applied to identify the defective scan cells. In [HUA03], algorithms are proposed to diagnose realistic chain defects by targeting intermittent scan chain faults. Note that with the more popular embedded compression, chain diagnosis procedures need to be enhanced to incorporate the compactor functions [HUA05]. Compared with the hardware-based methods, software-based techniques are more attractive due to not requiring design modification and no extra hardware overhead.

A typical software-based chain diagnosis algorithm, like [STA01] [GUO01] or [HUA03], is based on simulation. The chain diagnosis algorithm proposed in [STA01] includes two steps. Next, we will use an example to illustrate these two steps. Assume a design has one defective scan chain composed of 12 scan cells numbered from cell 0 to cell 11. Cell 0 is connected to scan chain output while cell 11 is connected to scan chain input.

Step 1: Identify faulty chains and fault types using chain integrity test patterns.

Suppose the faulty scan chain with 12 scan cells is loaded with a chain pattern 001100110011, where the leftmost bit is loaded into cell 11 and the rightmost bit is loaded into cell 0. If failures are observed at cells 2, 3, 6, 7, 10, 11 of this chain, then it can be determined that there is at least one stuck-at-1 fault on this chain. In reality, the chain defects can be modeled with various fault models such as stuck-at or hold-time fault.

Step 2: Locate faulty scan cells by injecting fault at each cell on the fault chain and simulating the failed scan patterns.

The objective of this step is to locate the suspect scan cells. If a fault is "injected" on a scan cell on the faulty chain, loaded values in the downstream of this scan cell on the faulty chain will be modified for all failed scan patterns. For example, suppose a scan pattern has good machine loaded value 001110011010 on the faulty chain. If a stuck-at-1 fault is injected on scan cell 8 of this chain, the loaded values will be modified as 001111111111. After pulsing the capture clock, simulated captured values in the upstream of the faulty scan cell on this chain will be modified. For example if the simulated captured value is 101011101011, the unloaded values will be 111111101011. The fault simulation is performed one cell at a time. The simulation results are compared with the results observed on ATE and the cell(s) that matches the best are finally reported as suspect(s).

Obviously, step 2 of the above-mentioned algorithm will need simulation of injecting fault on every scan cell on the whole faulty chain. If a faulty chain has a large number of scan cells, the run time could be prohibitive. To enhance the diagnosis efficiency, in [GUO01], an additional step is added between the above-mentioned two steps. This extra step will identify the range of suspect faulty cells before simulation.

In [GUO01], a full-masked method was proposed. In this method, each scan pattern is modified by changing the loaded values of all faulty scan chains to "X"s. Good machine simulation is performed with these modified patterns. If after simulating a pattern, some known values are captured into a faulty chain then bounds will be determined based on the captured known values and the corresponding tester observed values. In the example described earlier, assume a permanent stuck-at-1 fault exists on the defective scan chain with 12 scan cells. Suppose a scan test pattern loads 011000110001 into the scan cells on this faulty chain. Based on full-masked method, the loaded values for this faulty chain are changed to "XXXXXXXXXXXX". After good machine simulation, assume the captured values on the faulty chain are X10XX01X10XX. Since the loaded values for all faulty chains were changed to "X"s, the known values ("1"s and "0"s) captured into the faulty scan chain must be independent of the loaded values in the faulty chains. If the observed unloaded value at scan cell 9 is incorrect (i.e., simulated captured value is "0", ATE measured value is a "1"), a stuck-at-1 fault must be in the downstream of scan cell 9. In other words, cell 9 is upper bound (denoted UB thereafter) of the faulty cells. Similarly, if the observed unloaded value at scan cell 6 is correct (simulated capture value is "0", ATE measured value is "0"), the stuck-at-1 fault must be in the upstream of cell 6. In other words, cell 6 is lower bound (denoted LB thereafter) of the faulty cells. So the range of the stuck-at-1 fault is in [6, 9]. Hence, instead of simulating all 12 scan cells as in [STA01], the diagnosis algorithm proposed in [GUO01] will only have to simulate 4 scan cells. The run time will speedup 3X in this example.

In [HUA03], an iterative partial-masked method was proposed to enhance the range calculation in [GUO01]. The method initializes UB and LB to the leftmost and the rightmost scan cell of the faulty chain respectively. In each iteration, simulation is performed with the modified scan patterns and UB / LB are refined. The scan patterns are modified such that, instead of changing all the loaded values of the faulty chains to Xs like in the full-masked method, only the loaded values of scan cells from UB to right-most cell (cell 0) are modified to Xs. Consider the previous example, after determining that the fault is in the down stream of scan cell 9, loaded values of all scan patterns on the faulty chain are changed to "X" only from cell 9 to cell 0. This way, chances are increased that more known values are captured into the faulty chain in the downstream of cell 9 and more useful information is obtained for chain diagnosis. Once UB / LB is updated, one more iteration of partial-masked range calculation is performed by using the updated UB / LB. The above iterative procedure stops when the UB and LB cannot be updated anymore. It is possible that the range obtained by partial-masked method is further narrowed down and the simulation run time is further reduced compared to the full-masked method.

In [KUO06], parallel fault simulation is proposed. It simulates 32/64 chain faults for each pattern. However, it is unclear if this method can achieve better run time compared to parallel pattern simulation used in [GUO01][HUA03], where each simulation run 32/64 patterns with one injected chain fault.

3. Dynamic learning based chain diagnosis

In our chain diagnosis practice, we found that for big industrial designs, the diagnosis run rime could be unacceptably long, even after applying the previously mentioned partial-masked method to calculate range. In order to find the run time bottleneck, we select an industrial design to run some experiments. The design has about 2M gates and 64 scan chains. We select one longest scan chain that has 3181 scan cells. Each time, we inject one stuck-at-1 fault at a scan cell on this chain to create one fail log. The fault locations we selected are 6 cells apart along this chain. I.e., we select cells 0, 6, 12 … 3180. This way, we create 531 simulated chain failures cases. Due to ATE fail buffer size and test time limit, normally we cannot log too many failure patterns if a scan chain has defect. In this experiment, we only use the first 10 scan patterns to run chain diagnosis. We run partial-masked chain diagnosis algorithm, and statistically measure the run time spent in each step. We find that the time spend in the first step (identify faulty chain and fault type) is less than 1%. The time spent in the second step (partial-mask based range calculation) is less than 4%. The last step (simulation on each cell within the range) cost more than 95% of run time. Not surprisingly, the run time bottleneck is in the last simulation step. The reason behind this statistical result is that in many cases, we find range calculation cannot render a tight range such that the last step simulation has to be run on a large number of cells within the range. Statistically analysis shows that 201 out of 531 cases have range with more than 1000 scan cells. Evidently, partial-masked method based range calculation is still inefficient for many cases.

In the last step of chain fault simulation, a mismatch is defined as the simulated unloaded value at a scan cell or a PO is “1” (or “0”), whereas the ATE observed value at this bit is “0” (or “1”). If a fault is injected at a scan cell such that no mismatch happened, this cell is called a perfect-match cell. There could be one or multiple scan cells that are perfect-match cells. If there are multiple of them, it means the current patterns used for diagnosis cannot distinguish them. The real defect should be one of these perfect-match cells. Therefore, the diagnosis will try to find all perfect-match cells and report them as suspects. In the previous chain diagnosis algorithms, even we already find one perfect-match cell, we still have to continue run simulation on all other cells in the pre-calculated range. One interesting observation is that it is very likely that the perfect-match cells are neighboring cells within a segment of the faulty chain. That is to say, the indistinguishable chain faults are most likely on consecutive cells. Intuitively, this is easy to understand because the closer two faults are on a chain, the smaller probability that it creates enough different loaded and unloaded bits to distinguish the two faults. Hence the last step of simulation of chain fault at each cell within the range can be modified to finding the leftmost and rightmost perfect-match cell. Assuming the scan chain shift direction is from left to right, we call the leftmost perfect-match cell the perfect-match upper bound (denoted UB_p thereafter) and we call the rightmost perfect-match cell the perfect-match lower bound (denoted LB_p thereafter). All the cells within $[LB_p,UB_p]$ will not be simulated. Instead, we will report them as suspect as well. It is possible that one or more cells within $[LB_p , UB_p]$ are not perfect-match cells. We do not simulate each cell within $[LB_p , UB_p]$ due to two other reasons besides the run time efficiency.
(1) Statistical analysis of 531 chain failure cases showed that less than 2% of cases have nonconsecutive perfect-match cells.
(2) Such cases usually have too many perfect-match cells that make the diagnosis results having small value to guide PFA. Normally PFA only performed on the cases with less than 3 or 4 suspect cells.

Given the above explanations, now the last step of our new chain diagnosis algorithm is modified to finding $[LB_p , UB_p]$ from [LB , UB]. In the previous chain diagnosis algorithms, when we run fault simulation on a scan cell, and if the simulation does not match the ATE observation, we will discard the simulation results and try the next scan cell. In fact, we can learn some useful information from the previous unsuccessful simulation and use the learned information to direct us to tighten the range. Since each time we inject a chain fault at a different cell, we will learn different information from each simulation. We call this technique “dynamic learning”. If we only consider single permanent chain fault, there are five basic learning rules described as follows. We will use stuck-at fault as examples to illustrate each rule.

Rule 1: Inject a fault at the current LB, if mismatch happens at good chain or PO or on faulty chain at a cell < LB, back trace from the mismatched bit to the faulty chain. Update the LB to the cell LB_{new}, where LB_{new} is calculated as in the following steps:

Step 1: Find all cells on the faulty chain that are driving the mismatched bit. Put these driving cells into a set *C*.

Step 2: For the mismatched pattern, if the good machine loaded value at a driving cell is not sensitive to the chain fault (i.e., the chain fault has no impact to its loaded value), remove this driving cell from *C*.

Step 3: If a driving cell is in the downstream of LB (i.e. its cell number < LB), drop this cell from *C*.

Step 4: Set LB_{new} as the cell with the minimum cell number in *C*.

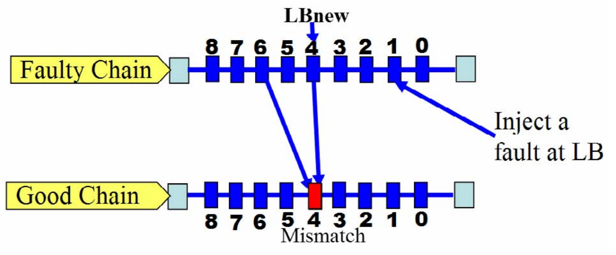

Figure 1 An Example for Rule 1

An example to explain Rule 1 is illustrated in Figure 1. Suppose the current LB is cell 1 on the faulty chain. Assume if we inject a fault at cell 1 on the faulty chain, we find a mismatch happens on cell 4 on a good chain (highlighted in red color). Obviously the mismatch on any good chain or PO or on faulty chain at a cell < LB must be caused by the mismatch between the simulated loaded values at the faulty chain and their real loaded values on ATE. By using critical path tracing, we can find all cells on the faulty chain that are driving the mismatched bit. Assuming cells 0, 2, 4, 6 on the faulty chain are driving cell 4 on the good chain, we initialize $C = \{0, 2, 4, 6\}$ in step 1. By considering the pattern loaded value and the fault type, we can identify which cells will not be impacted by the chain fault. In this example, suppose cell 2 on the faulty chain has a good machine loaded value "1" and we know the faulty chain has stuck-at-1 fault. We can deduce that the mismatch must not be caused by cell 2. That is to say, at least one mismatched loaded value at cells 0, 4, 6 at the faulty chain caused the mismatched simulation result at cell 4 on the good chain. Therefore we get $C = \{0, 4, 6\}$ in step 2. Since cell 0 is in the downstream of the currently injected fault (cell 1), cell 0's loaded values has been modified by considering the chain fault. That left us only cells 4 and 6 that may have caused the mismatch, which leads us to update $C = \{4, 6\}$ in step 3. Finally in step 4, to be conservative, we set the minimum cell number (cell 4) in C as the updated lower bound LB_{new}. This way, the learned information will direct us to inject the next fault at cell 4 instead of cell 2 as in the traditional diagnosis algorithm.

Rule 1 Extension:

When we apply Rule 1 mentioned above, if we find multiple mismatched bits at good chains or POs or on faulty chain at a cell < LB, we apply Rule 1 to every mismatched bit i and get $LB_{new,i}$. Finally we pick $LB_{new} = \text{Maximum}(LB_{new,\,i})$. Proof Skipped.

Rule 2: Inject a fault at the current LB, if mismatch happens on the faulty chain at a cell ≥ LB. Update the LB to the cell LB_{new} in the following 3 steps:

Step 1: Find all mismatched cells ≥ LB on the faulty chain. Put these cells into a set C.

Step 2: Pick from C the cell with the maximum id, and denote its cell number as M.

Step 3: Set $LB_{new} = M + 1$

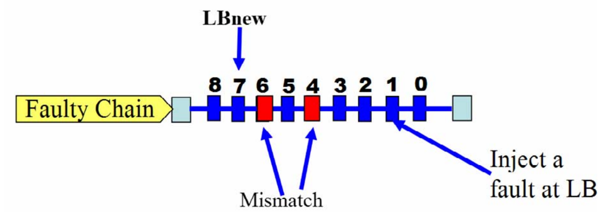

Figure 2 An Example for Rule 2

An example to explain Rule 2 is illustrated in Figure 2. Suppose the current LB is cell 1 on the faulty chain. Assume if we inject a stuck-at-1 fault at cell 1 on the faulty chain, we find two mismatches happening on cells 4 and 6 on the faulty chain. It implies:

(1) The simulated value at these two cells must be "1" since the injected stuck-at-1 fault at cell 1 will change their unloaded values to "1".

(2) The real ATE observations at these two cells must be "0".

We can learn that the fault should not block the unloading procedure of cells 4 and 6. Therefore, we initialize $C = \{4, 6\}$ in step 1, and set $M = 6$ in step 2. Finally, we move LB_{new} to cell $6+1=7$. Note that this rule applies only to permanent chain fault but not for intermittent chain fault.

Rule 3: Inject a fault at the current UB, if mismatch happens on good chain or PO or on faulty chain at a cell < LB, back trace from the mismatched point to the faulty chain. Update the UB to the cell UB_{new}, where UB_{new} is calculated as in the following steps:

Step 1: Find all cells on the faulty chain that are driving the mismatched bit. Put these driving cells into a set C.

Step 2: For the mismatched pattern, if the good machine loaded value at a driving cell is not sensitive to the chain fault (i.e., the chain fault has no impact to its loaded value), remove this driving cell from C.

Step 3: If a driving cell is in the upstream of UB, drop this driving cell from C.

Step 4: Pick from C the cell with the maximum cell number, and denote its cell number as M.

Step 5: Set $UB_{new} = M - 1$

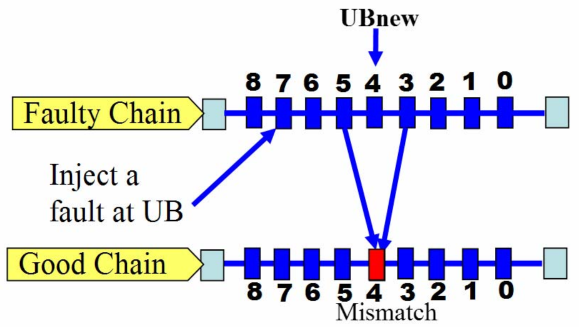

Figure 3 An Example for Rule 3

An example to explain Rule 3 is illustrated in Figure 3. Suppose the current UB is cell 7 on the faulty chain. Assume if we inject a fault at cell 7 on the faulty chain, we find a mismatch happens on cell 4 on a good chain (highlighted in red color). Obviously the mismatch at any good chain or PO or on faulty chain at a cell < LB, must be caused by the mismatch between the simulated loaded values at the faulty chain and their real loaded values on ATE. By using critical path tracing, we can find all cells on the faulty chain that are driving the mismatched bit. Assuming cells 1, 3, 5, 8 on the faulty chain are driving cell 4 on the good chain, we initialize $C = \{1, 3, 5, 8\}$ in step 1. By considering the pattern loaded value and the fault type, we can identify which cells will not be impacted by the chain fault. In this example, suppose cell 1 on the faulty chain has a good machine loaded value "1", and we know the faulty chain has stuck-at-1 fault. We can deduce that the mismatch must not be caused by cell 1 on the faulty chain. Hence, we remove cell 1 from C and get $C = \{3, 5, 8\}$ in step 2. Since cell 8 is in the upstream of the currently injected fault (cell 7), its loaded values are not impacted by the fault. So we remove cell 8 from C and get $C = \{3, 5\}$ in step 3. At this point, we can learn that the chain fault should have no impact on at least one of the cells in C. To be conservative, in step 4, we pick the cell with the maximum cell number in C and set $M = 5$. Finally, we set the updated upper bound $UB_{new} = 5\text{-}1 = 4$. This way, the learned information will direct us to inject the next fault at cell 4 instead of cell 6 as in the traditional diagnosis algorithm.

Rule 3 Extension:

When we apply Rule 3 mentioned above, if we find multiple mismatched bits at good chains or POs or on faulty chain with mismatched cell < LB, we apply the learning rule to every mismatched bit i and get $UB_{new,i}$. Finally we set UB_{new}=Minimum($UB_{new,\ i}$). Proof Skipped.

Rule 4: Inject a fault at the current UB, suppose a mismatch satisfies the following 2 conditions simultaneously:

(1) The mismatch happens on the faulty chain at a cell such that UB < cell < LB.

(2) The ATE observed value at this mismatched cell is **inconsistent** with the fault type of the faulty chain.

We can apply both learning Rules 2 and 3 (and its extension) introduced above.

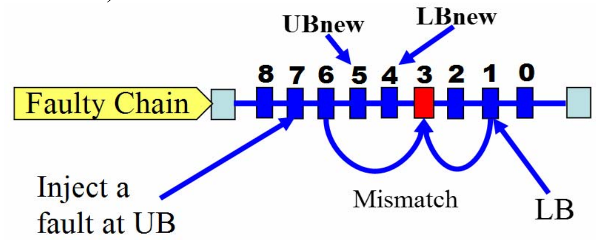

Figure 4 An Example for Rule 4

An example to explain Rule 4 is illustrated in Figure 4. Suppose the faulty chain has current UB at cell 7 and LB at cell 1. Assume if we inject a fault at cell 7 on the faulty chain, we find a mismatch happens on cell 3 on the same faulty chain (highlighted in red color). If the faulty chain has a stuck-at-1 fault, and ATE observation on cell 3 is "0", it implies the simulated unload value at cell 3 is "1". We know that there should be no stuck-at-1 fault in the downstream of cell 3. Otherwise, we cannot observe "1" at cell 3 on ATE. So Rule 2 can be applied to update LB_{new}= 4. We also deduce that the simulated "1" at cell 3 are caused by the mismatched driving bits. So we can apply Rule 3 as well. Assuming cell 1 and cell 6 are the driving cells of cell 3, we will update the UB_{new}=5 per Rule 3. Note that this learning rule can update both UB and LB.

Rule 5: Inject a fault at the current UB, suppose a mismatch satisfies the following 2 conditions simultaneously:

(1) The mismatch happens on the faulty chain at a cell such that UB < cell < LB. (Same condition as in Rule 4)

(2) The ATE observed value at this mismatched cell is **consistent** with the fault type of the faulty chain.

We can only apply learning Rule 3 and its extension introduced above. However, we cannot apply Rule 2. Still using the previous example illustrated in Figure 4, but we change the chain fault type to stuck-at-0 this time. If the faulty chain has a stuck-at-0 fault, and ATE observation on cell 3 is "0", it implies the simulated unload value at cell 3 is "1". This time we do not know whether there is a stuck-at-0 fault in the downstream of cell 3 or not. So Rule 2 cannot be applied. Rule 3 can still be applied.

If any learning rule is applicable during searching of LB_{new}/UB_{new}, using the learning rules will speedup the searching compared to the traditional chain diagnosis algorithm that searches all cells in the range one by one. By learning, the next simulated cell may not be the neighbor cell of the currently simulated cell. The "jump distance" depends on the mismatched patterns, mismatched scan cell locations and the circuit structure. If no rule is applicable at some point, using next-cell searching, i.e., LB_{new}= LB+1 or UB_{new} = UB-1, may make learning rules applicable at the new searching points. We start searching from current UB (LB) and stop searching if we find the perfect-match cell UB_p (LB_p).

Note that so far we only illustrated the basic dynamic learning rules by considering (1) single faulty chain, (2) single fault per chain, (3) permanent chain fault, (4) non-compactor based scan architecture and (5) stuck-at fault as examples. In reality we have to consider (1) multiple faulty chains, (2) multiple faults (with same fault model) per chain, (3) intermittent chain faults, (4) scan architectures with any embedded compactor and (5) any fault models including timing-related chain fault (e.g. scan chain hold-time fault). The proposed dynamic learning can be implemented by either adapting existing rules or adding new rules. Due to space limit, we do not illustrate more complicated learning techniques to handle these complicated cases. We already implemented all rules to consider all those real case situations into a commercial diagnosis tool.

4. Experimental results

In order to measure the efficiency of the proposed dynamic-learning based chain diagnosis, we compare one old chain diagnosis and the new learning based chain diagnosis algorithm on 531 simulated chain failures cases mentioned in Section 3. The old chain diagnosis we used for comparison is the one proposed in [HUA03] since it is more efficient than the algorithms introduced in [STA01] and [GUO01]. As introduced in Section 2, this chain diagnosis algorithm used partial-masked range calculation and simulation of each scan cell within the range. The run time speedup ratios are shown in Figure 5.

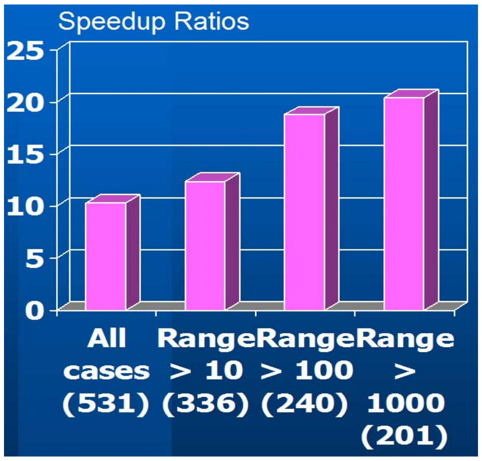

Figure 5 Run Time Speedup Ratios

We measure run time speedup ratios for all 531 test cases and calculate the average. The proposed dynamic-learning based diagnosis algorithm is about 10X enhancement, which is shown as the leftmost bar in Figure 5. In the rest of bars we show the speedup ratios for only a subset of circuits. Among 531 cases, 336 of them had partial-masked range > 10. Among these 336 cases, 240 of them had partial-masked range > 100. Among 240 cases, 201 of them had partial-masked range > 1000. The experimental results indicate that the speedup ratios are about 12X, 18X and 20X for the subsets of cases with ranges >10, >100 and >1000 respectively. Obviously, the larger a range is, the more efficient the proposed dynamic-learning based chain diagnosis shows. In terms of absolute run time, the proposed dynamic learning based chain diagnosis use only about a few minutes on average for one case. The diagnosis experiments are run on a 64-bits Linux machine.

Diagnosis quality of results are measured in terms of their accuracy (whether the reported suspect(s) include the injected fault) and resolution (the number of suspects reported). In all of the 531 cases, accuracy was not a problem. The resolution is almost the same the old chain diagnosis algorithms.

5. Conclusions

In this paper, we investigated the run time bottleneck for the traditional chain diagnosis algorithms. A novel dynamic-learning based chain diagnosis methodology is proposed. The new algorithm is based on several learning rules. The rules analyze the circuit, patterns, and mismatched bits to figure out what cell(s) should be simulated in the next iteration. Therefore instead of simulating every one cell within a range, we may only need simulation of a few cells to find out perfect-match suspects. Experiments with a large number of simulated test cases showed that the proposed methods can achieve about 10X run time speedup on average. Although we discuss learning based chain diagnosis in this paper, the concept may also be applied to system logic diagnosis if a new set of learning rules are defined for logic diagnosis.

REFERENCES

[CHE04] W.-T. Cheng, K.-H. Tsai, Y. Huang, N. Tamarapalli and J. Rajski, “Compactor Independent Direct Diagnosis,” Asian Test Symposium, 2004, pp. 204 – 209.
[CRO05] A. Crouch, “Debugging and Diagnosing Scan Chains,” EDFAS, Vol. 7, Feb., 2005, pp 16-24.
[EDI95] S. Edirisooriya and G. Edirisooriya, “Diagnosis of Scan Failures,” Proc. VLSI Test Symposium 1995, pp.250-255.
[GUO01] R. Guo and S. Venkataraman, “A Technique for Fault Diagnosis of Defects in Scan Chains,” Proc. Int’l Test Conference, 2001, pp. 268-277.
[HUA03] Y. Huang, W.-T. Cheng, S.M. Reddy, C.-J. Hsieh, Y.-T. Hung, “Statistical Diagnosis for Intermittent Scan Chain Hold-Time Fault,” Int’l Test Conference, 2003, pp.319-328.
[HUA05] Y. Huang, W.-T. Cheng, and J. Rajski, “Compressed Pattern Diagnosis For Scan Chain Failures,” Proc. Int’l Test Conf. 2005.
[KUN94] S. Kundu, “Diagnosing Scan Chain Faults,” IEEE Trans. On VLSI Systems, Vol. 2, No. 4, 1994, pp.512-516.
[KUO06] Y.-L. Kuo, W.-S. Chuang and J. C.-M. Li, "Jump simulation: a technique for fast and precise scan chain fault diagnosis," Proc. Int'l Test Conference, 2006, Paper 22.1.
[NAR97] S. Narayanan and A. Das, “An Efficient Scheme to Diagnose Scan Chains,” Proc. Int’l Test Conference, 1997, pp. 704-713.
[SCH92] J. Schafer, F. Policastri and R. Mcnulty, “Partner SRLs for Improved Shift Register Diagnostics,” Proc. VLSI Test Symposium, 1992, pp.198-201.
[STA01] K. Stanley, “High-Accuracy Flush-and-Scan Software Diagnostic,” IEEE Design & Test of Computers, Nov-Dec, 2001, pp.56-62.
[WU98] Y. Wu, “Diagnosis of Scan Chain Failures,” Proc. Int’l Symp. on Defect and Fault Tolerance in VLSI Systems, 1998, pp.217-222.