RT-Level Deviation-Based Grading of Functional Test Sequences*

Hongxia Fang[1], Krishnendu Chakrabarty[1], Abhijit Jas[2], Srinivas Patil[2] and Chandra Tirumurti[2]

[1]ECE Dept., Duke University, Durham, NC
{hf12, krish}@ee.duke.edu

[2]Intel Corporation, Austin, TX
{abhijit.jas, srinivas.patil, chandra.tirumurti}@intel.com

***Abstract*—Functional test sequences are often used in manufacturing testing to target defects that are not detected by structural test. Therefore, it is necessary to evaluate the quality of functional test sequences. However, it is very time-consuming to evaluate the quality of functional test sequences by gate-level fault simulation. Therefore, we propose output deviations as a metric to grade functional test sequences at the register transfer (RT)-level without explicit fault simulation. Experimental results for the open-source Parwan processor and the Scheduler module of the Illinois Verilog Model (IVM) show that the deviations metric is computationally efficient and it correlates well with gate-level coverage for stuck-at, transition-delay, and bridging faults. Results also show that functional test sequences that are reordered based on output deviations provide steeper gate-level fault coverage ramp-up compared to other ordering methods.**

I. Introduction

Structural test offers several advantages over functional test. For example, it allows us to develop test generation and test evaluation (e.g., fault simulation) algorithms based on selected fault models [1]. However, functional test has not yet been completely replaced by structural test. Instead, it is commonly used in industry to target defects that are not detected by structural tests [2]–[4]. An advantage of functional test is that it avoids overtesting since it is performed in normal functional mode. In contrast, structural test is accompanied by some degree of yield loss [5]. Register transfer (RT)-level fault modeling, test generation and test evaluation are therefore of considerable interest [6]–[9].

Given a large pool of functional test sequences (for example, design verification sequences), it is necessary to develop an efficient method to select a subset of sequences for manufacturing testing. Since functional test sequences are much longer than structural tests, it is time-consuming to grade functional test sequences using traditional gate-level fault simulation methods.

To quickly estimate the quality of functional tests, a high-level coverage metric for estimating the gate-level coverage of functional tests is proposed in [10]. This metric is based on event monitoring. First, gate-level fault activation and propagation conditions are translated to coverage objects at functional level. Next, a functional simulator is used for monitoring the "hit number" of the coverage objects, and estimating the fault coverage. However, this approach requires considerable time and resources for the extraction of the coverage objects. In particular, experienced engineers and manual techniques are needed to extract the best coverage objects.

In [11], another coverage metric is proposed to estimate the gate-level fault coverage of functional test sequences. This metric is based on logic simulation of the gate-level circuit, and on the set of states that the circuit traverses. It relies on the observation that a test sequence with high fault coverage also traverses a large number of circuit states [12] [13]. The ratio of visited states to the total number of possible states for each subset is used to estimate fault coverage. However, this method does not take the observability of flip-flops into consideration. The observability of flip-flops affects the accuracy of the estimated fault coverage, especially when state transitions in the circuit are caused by changes in the contents of sequentially-deep flip-flops. Another drawback of this method is that it is impractical for large designs since it requires gate-level logic simulation.

In this paper, we propose output deviations as a metric at RT-level to grade functional test sequences. The deviation metric at the gate-level has been used in [14] to select effective test patterns from a large repository of n-detect test patterns. It has also been used in [15] to select appropriate LFSR seeds for LFSR-reseeding-based test compression. Here, we define the output deviations metric at RT-level and use it for grading functional test sequences. We show that, compared to gate-level fault simulation, an order of magnitude reduction in computation time is achieved using the proposed method.

The remainder of this paper is organized as follows. Section II introduces basic concepts and preliminaries. Section III defines output deviations and describes the procedure used for calculation. We present the experimental results on the open-source Parwan processor and the Scheduler module in Section IV. Section V concludes the paper.

II. Output Deviations at RT-level: Preliminaries

In this section, we present basic concepts needed to calculate output deviations at RT-level. Our objective is to use deviations as a surrogate metric for functional test grading.

First, we define the concept of transition count (TC) for a register. Typically, there is dataflow between registers when an instruction is executed and the dataflow affects the values of registers. For any given bit of a register, if the dataflow causes a change from 0 to 1, we record that there is a $0 \rightarrow 1$ transition. Similarly, if the dataflow causes a change from 1 to 0, we record that there is a $1 \rightarrow 0$ transition. If the dataflow makes no change to this bit of the register, we record that there is a $0 \rightarrow 0$ transition or a $1 \rightarrow 1$ transition, as the case may be.

After a transition occurs, the value of the bit of a register can be correct or faulty (due to an error). With any transition of a register bit, we associate a "confidence level" (CL) parameter, which represents the probability that the correct transition occurs. The CL is not associated with the test sequence; rather, it provides a probabilistic measure of the correct operation of instructions at the RT-level. It can be estimated from low-level failure data or it can be provided by the designer based on the types of faults of interest. Low CL values can be assigned to registers that are to be especially targeted by functional test

*This work was supported in part by the Semiconductor Research Corporation under Contract no. 1588, and by an equipment grant from Intel Corporation.

1093-0167/09 $25.00 © 2009 IEEE
DOI 10.1109/VTS.2009.12

sequences for error observation and propagation. We show later that small differences in the CL values have little impact on the effectiveness of the proposed method for grading functional test sequences.

Without loss of generality, we assume there that the CL values for 0→0 and 1→1 are higher than that for the transitions 0→1 and 1→0 for a register bit. We also assume that the CL values for 0→1 and 1→0 are identical. The CL for a register can be described as a 4-tuple, e.g., $< 0.998, 0.995, 0.995, 0.998 >$, where the elements in the tuple correspond to the transitions 0→0, 0→1, 1→0, and 1→1, respectively. While different CL values can be used for the various registers in a design, we assume here without loss of generality that all registers have identical CL values.

When an instruction is executed, there may be several transitions for the bits of a register. Therefore, an error may be manifested after the instruction is executed. We define the CL for instruction I_i, C_i, as the probability that no error is produced when I_i is executed. Its calculation is explained in Section III.

Similarly, since a functional test sequence is composed of several instructions, we define the CL for a functional test sequence to be the probability that no error is observed when this functional test sequence is executed. For example, suppose a functional test sequence, labeled T_1, is composed of instructions I_1, I_2, ..., I_N, and let C_i be the CL for I_i, as defined above. The CL value for T_1, $C(T_1)$, is defined as: $C(T_1) = \prod_{i=1}^{N} C_i$. This corresponds to the probability that no error is produced when T_1 is executed. We define the deviation for functional test sequence T_1, $\triangle(T_1)$, as $1 - C(T_1)$, i.e., $\triangle(T_1) = 1 - \prod_{i=1}^{N} C_i$.

Based on these definitions, output deviations can be calculated at RT-level for functional test sequences for a given design. This procedure is described in detail in the next section.

III. Deviation Calculation at RT-level

Since our objective is to use deviations as a surrogate metric to grade functional test sequences, we expect test sequences with higher deviation values to provide higher defect coverage. In order to ensure this, we consider three contributors to output deviations. The first is the TC of registers. Higher the TC for a functional test sequence, the more likely is it that this functional test sequence will detect defects. We therefore take TC into consideration while calculating deviations. The second contributor is the observability of a register. The TC of a register will have little impact on defect detection if its observability is so low that transitions cannot be propagated to primary outputs. The third contributor is the amount of logic connected to a register. In order to have a high correlation between RT-level deviation and gate-level stuck-at fault coverage, we need to consider the relationship between RT-level registers and gate-level components.

In this section, the Parwan processor [16] is used as an example to illustrate the calculation of output deviations.

A. Observability Vector

We first consider the observability of registers. The output of each register is assigned an observability value. The observability vector for a design at RT-level is composed of the observability values of all its registers. Let us consider the calculation of the observability vector for the Parwan processor [16]. The Parwan is an accumulator-based 8-bit processor with a 12-bit address bus. Its architectural block diagram is shown in Figure 1(a).

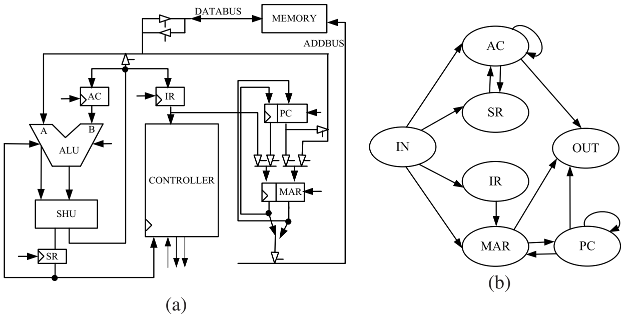

Figure 1. (a) Architecture of the Parwan processor; (b) Dataflow graph of the Parwan processor (only the registers are shown).

From the instruction-set architecture and RT-level description of Parwan, we extract the dataflow diagram to represent all possible functional paths. Figure 1(b) shows the dataflow graph of Parwan. Each node represents a register. The IN and OUT nodes represent memory. A directed edge between registers represents a possible functional path between registers. For example, there is an edge between the AC node and the OUT node. This edge indicates that there exists a possible functional path from register AC to memory.

From the dataflow diagram, we can calculate the observability vector. First, we define the observability value for the OUT node. The primary output OUT has the highest observability since it is directly observable. Using sequential-depth-like measure for observability [1], we define the observability value of OUT to be 0, written as $OUT\_obs = 0$. For every other register node, we define its observability parameter as 1 plus the minimum of the observability parameters of all its fanout nodes. For example, the fanout nodes of register AC are OUT, SR, and AC itself. Thus the observability parameter of register AC is 1 plus the minimal observability parameter among OUT, SR, AC. That is, the observability parameter of register AC is 1. In the same way, we can obtain the observability parameters for MAR, PC, IR and SR. We define the observability value of a register as the reciprocal of its observability parameter. Finally, we obtain the observability vector for Parwan. It is simply $(\frac{1}{AC\_obs}, \frac{1}{IR\_obs}, \frac{1}{PC\_obs}, \frac{1}{MAR\_obs}, \frac{1}{SR\_obs})$, i.e., $(1, 0.5, 1, 1, 0.5)$.

B. Weight Vector

The weight vector is used to model how much combinational logic a register is connected to. Each register is assigned a weight value, representing the relative sizes of its input cone and fanout cone. The weight vector for a design is composed of the weight values of all its registers. Obviously, if a register has a large input cone and a large fanout cone, it will affect and be affected by many lines and gates. Thus it is expected to contribute more to defect detection. In order to accurately extract this information, we need gate-level information to calculate the weight vector. We only need to report the number of stuck-at faults for each component based on the gate-level netlist. This can be easily implemented without gate-level logic simulation by an automatic test pattern generation (ATPG) tool or a design analysis tool. Here we use Flextest to obtain the number of stuck-at faults for

TABLE I
WEIGHT VECTOR FOR REGISTERS (PARWAN).

	No. of FAR	Weight value
AC	2172	1
IR	338	0.1556
PC	936	0.4309
MAR	202	0.093
SR	2020	0.930

TABLE II
TCS FOR TS.

	0→0	0→1	1→0	1→1
AC	67	61	60	44
IR	206	67	66	37
PC	708	90	85	205
MAR	913	251	246	246
SR	67	11	14	12

TABLE III
EFFECTIVE TCS FOR TS.

Register ID	Register name	0→0	0→1	1→0	1→1
R_1	AC	67	61	60	44
R_2	IR	16.03	5.21	5.13	2.88
R_3	PC	305.08	38.78	36.63	88.33
R_4	MAR	84.91	23.34	22.88	22.88
R_5	SR	31.16	5.115	6.51	5.58
Sum		504.18	133.45	131.19	163.67

each component in Parwan: there are 248 stuck-at faults in AC, 136 in IR, 936 in PC, 202 in MAR, 96 in SR, 14690 in ALU, and 464 in SHU. From this information, we can easily calculate the weight vector (Table I).

Based on the RT-level description of the design, we can determine the fanin cone and fanout cone of a register. For example, the AC register is connected to three components: AC, SHU and ALU. Given a set of registers $\{R_i\}, i = 1, 2, .., n$, let f_i be the total number of stuck-at faults in components connected to register R_i. Let $f_{max} = \max\{f_1, .., f_n\}$. We define the weight of register R_i as f_i/f_{max} to normalize the size of gate-level logic. Table I shows the numbers of faults affecting registers and weights of registers. We can see that $f_{max} = 2172$ for Parwan processor, which is the number of faults in AC. The weight of IR can be calculated as 338/2172, i.e., 0.1556. In this way, weights of other registers can also be obtained. Finally, we get the weight vector $(1, 0.1556, 0.4309, 0.093, 0.930)$.

C. Calculation of Output Deviations

The output deviations can be calculated for functional test sequences using the TCs, the observability vector, and the weight vector. First, we need to calculate the effective TCs for registers. Suppose register AC makes N_1 0→1 transitions for a functional test sequence TS. Then its effective 0→1 TC equals the product of N_1, the observability value of register AC, and the weight of register AC. The following example illustrates how to calculate the deviation for TS. Consider the CL vector $(1, 0.998, 0.998, 1)$, the observability vector $(1, 0.5, 1, 1, 0.5)$, and the weight vector $(1, 0.1556, 0.4309, 0.093, 0.930)$. Also, the TCs corresponding to TS are shown in Table II.

In Table II, each row shows the TC for one register, while each column represents the transition type. For example, the value of third row and second column is 206, which implies that the 0→0 TC for IR is 206 when functional test sequence TS is executed.

By considering the weight vector and the observability vector, the TC of a register can be transformed to the effective TC. Table III shows the effective TC values for TS for the given observability vector and weight vector.

In Table III, each row lists the effective TC for one register. The last row shows the aggregated effective TC for all registers. The columns indicate the various types of transitions.

Suppose TS is composed of 50 instructions I_1, I_2 ..., I_{50}. For each instruction I_i, suppose the effective 0→0 TC for register R_k (where R_k, $1 \leq k \leq 5$, is listed in Table III) is R_{ki00}, the effective 0→1 TC for register R_k is R_{ki01}, the effective 1→0 TC for register R_k is R_{ki10}, and the effective 1→1 TC for register R_k is R_{ki11}. Given the CL vector $(1, 0.98, 0.98, 1)$, the CL value C_i for instruction I_i can be calculated by considering all possible transitions and the different registers:

$$C_i = \prod_{k=1}^{5}(1^{R_{ki00}} \cdot 0.998^{R_{ki01}} \cdot 0.998^{R_{ki10}} \cdot 1^{R_{ki11}}). \tag{1}$$

Using the property of the exponents, whereby $x^a \cdot x^b = x^{a+b}$, Equation (1) can be rewritten as

$$C_i = 1^{\sum_{k=1}^{5} R_{ki00}} \cdot 0.998^{\sum_{k=1}^{5} R_{ki01}} \cdot \\ 0.998^{\sum_{k=1}^{5} R_{ki10}} \cdot 1^{\sum_{k=1}^{5} R_{ki11}}. \tag{2}$$

Let $S_{i00}=\sum_{k=1}^{5} R_{ki00}$, $S_{i01}=\sum_{k=1}^{5} R_{ki01}$, $S_{i10}=\sum_{k=1}^{5} R_{ki10}$, and $S_{i11}=\sum_{k=1}^{5} R_{ki11}$. Equation (2) can now be written as

$$C_i = 1^{S_{i00}} \cdot 0.998^{S_{i01}} \cdot 0.998^{S_{i10}} \cdot 1^{S_{i11}}. \tag{3}$$

Based on the deviation definition in Section II, the deviation for TS can be calculated as $\Delta(TS) = 1 - \prod_{i=1}^{50} C_i$, i.e.,

$$\Delta(TS) = 1 - \prod_{i=1}^{50}(1^{S_{i00}} \cdot 0.998^{S_{i01}} \cdot 0.998^{S_{i10}} \cdot 1^{S_{i11}}) \tag{4}$$
$$= 1 - 1^{\sum_{i=1}^{50} S_{i00}} \cdot 0.998^{\sum_{i=1}^{50} S_{i01}} \cdot 0.998^{\sum_{i=1}^{50} S_{i10}} \cdot 1^{\sum_{i=1}^{50} S_{i11}}$$

Let $S_{00}^* = \sum_{i=1}^{50} S_{i00}$, $S_{01}^* = \sum_{i=1}^{50} S_{i01}$, $S_{10}^* = \sum_{i=1}^{50} S_{i10}$, and $S_{11}^* = \sum_{i=1}^{50} S_{i11}$, Equation (4) can be rewritten as:

$$\Delta(TS) = 1 - 1^{S_{00}^*} \cdot 0.998^{S_{01}^*} \cdot 0.998^{S_{10}^*} \cdot 1^{S_{11}^*}.$$

Note that S_{00}^* is the aggregated effective 0→0 TC of all registers for all the instructions in TS. The parameters S_{01}^*, S_{10}^*, and S_{11}^* are defined in a similar way. From Table III, we can see that the aggregated effective TC is $(504.18, 133.45, 131.19, 163.67)$. Thus we have $\Delta(TS) = 1 - 1^{504.18} \cdot 0.998^{133.45} \cdot 0.998^{131.19} \cdot 1^{163.67}$, which implies that $\Delta(TS) = 0.4113$.

IV. Experimental Results

We next evaluate the efficiency of deviation-based test-sequence by performing experiments on the Parwan processor and the Scheduler module of the Illinois Verilog Model (IVM) [17] [18]. IVM employs a microarchitecture that is similar in complexity to the Alpha 21264. It has most of the features of modern microprocessors, featuring superscalar operation, dynamical scheduling, and out-of-order execution. Relevant research based on IVM has recently been reported in [19] [20]. The Verilog model for Scheduler consists of 1090 lines of code. A synthesized gate-level design for it consists of 375,061 gates and 8,590 flip-flops.

Our first goal is to show high correlation between RT-level deviations and gate-level coverage for various fault models. The second goal is to investigate the ramp-up of the gate-level fault coverage for various functional test sequences. These sequences are obtained using different sequence-reordering methods.

A. Results for Parwan Processor

1) Experimental setup: All experiments for Parwan processor were performed on a 64-bit Linux server with 4 GB memory. The program for deviation calculation was coded in C++. We used ModelSim to run the Verilog simulation. The Flextest tool was used to run gate-level fault simulation and Matlab

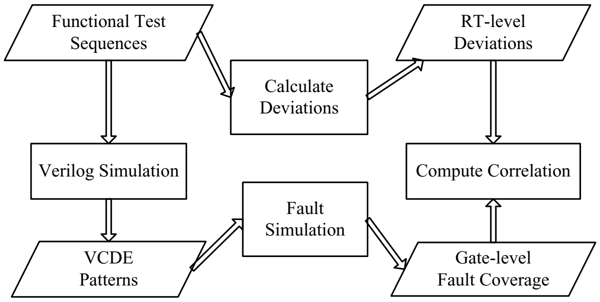

Figure 2. Experimental flow.

was used to obtain the Kendall's correlation coefficient [21]. The Kendall's correlation coefficient is used to measure the degree of correspondence between two rankings and assessing the significance of this correspondence. It is used in this paper to measure the degree of correlation of RT-level deviations and gate-level fault coverage. A coefficient of 1 indicates perfect correlation while a coefficient of 0 indicates no correlation.

We obtained a functional test program [22] for Parwan and divided it into ten test sequences TS_1, TS_2, ..., TS_{10}. Each sequence consists of 50 instructions. For example, TS_1 is composed of the first 50 instructions in the test program.

2) Correlation between output deviations and gate-level fault coverage: For these ten functional test sequences, we derived the Kendall's correlation coefficient between their RT-level output deviations and their gate-level fault coverage. The stuck-at fault model is used for the Parwan processor, as well as the bridging-coverage estimate (BCE) [23] and the modified BCE measure (BCE^+) [24]. Figure 2 presents details about the experimental flow. First we calculate the deviation for the ten functional test sequences under eight CL vectors. The CL vectors are chosen as follows: CL_1 = (1, 0.998, 0.998, 1), CL_2 = (0.99998, 0.995, 0.995, 0.99998), CL_3 = (0.99997, 0.9993, 0.9993, 0.99997), CL_4 = (1, 0.9992, 0.9992, 1), CL_5 = (1, 0.996, 0.996, 1), CL_6 = (0.9997, 0.994, 0.994, 0.9997), CL_7 = (0.99998, 0.9996, 0.9996, 0.99998), and CL_8 = (1, 0.9993, 0.9993, 1). For each CL vector, we obtain the deviation values for the ten functional test sequences and record them as a vector. For example, for the CL_1 case, the deviations are recorded as $DEV\_CL_1(dev_1, dev_2, ..., dev_{10})$.

We next obtain test patterns in external-VCD (VCDE) format for the ten functional test sequences by running Verilog simulation. Using the VCDE patterns, gate-level fault coverage is obtained by running fault simulation for the ten functional test sequences. The coverage values are recorded as a vector: $COV(cov_1, cov_2, ..., cov_{10})$. Table IV shows the gate-level fault coverage for the ten functional test sequences.

In order to evaluate the effectiveness of the deviation-based functional test-grading method, we calculate the Kendall's correlation coefficient between DEV_CL_i ($i = 1, ..., 8$) and COV for each CL vector. Figure 3 shows the correlation between deviations and stuck-at fault coverage, as well as between deviations and BCE/BCE^+ metrics, for different CL vectors. For stuck-at faults, we see that the coefficients are very close to 1 for all four CL vectors. For bridging faults, the correlation is less, but still significant. The results demonstrate that the deviation-based method is effective for grading functional test sequences. The CPU time for deviation calculation and test-sequence grading is less than 1 second in all cases.

TABLE IV
GATE-LEVEL FAULT COVERAGE (PARWAN).

Functional test	Stuck-at fault coverage (%)	BCE (%)	BCE^+ (%)
TS_1	60.50	54.61	40.18
TS_2	55.04	49.42	35.10
TS_3	55.71	50.40	36.19
TS_4	56.20	50.43	36.33
TS_5	54.47	52.20	37.97
TS_6	57.23	53.61	39.07
TS_7	59.99	55.20	40.63
TS_8	61.17	55.39	41.14
TS_9	61.17	56.01	41.50
TS_{10}	55.99	52.13	37.74

TABLE V
REGISTERS AND REGISTER ARRAYS IN THE SCHEDULER MODULE.

Name	Width	Array	Name	Width	Array
sb_counters	3	0 : 79	instout0	222	–
issued	1	0 : 31	instout1	222	–
valid	1	0 : 31	instout2	222	–
issue_head	5	–	instout3	222	–
issue_tail	5	–	instout4	222	–
inst_array	217	0 : 31	instout5	222	–

B. Results for Scheduler module

To further demonstrate the benefits of the RT-level deviation metric for larger circuits and longer test sequences, we perform experiments on the Scheduler module.

The experimental procedure is similar to that of Parwan processor: 1) obtain the RT-level deviations by considering TCs, observability vector, and weight vector; 2) obtain the gate-level fault coverage; 3) calculate the correlation coefficient and compare the coverage ramp-up curves.

1) Scheduler module: The Scheduler dynamically schedules instructions, and it is a key component of the IVM architecture [19]. It contains an array of up to 32 instructions waiting to be issued and can issue 6 instructions in each clock cycle.

Table V shows the registers and register arrays in the Scheduler module. The first and the fourth columns list the name of the register or register array. The second and fifth columns represent the width of each register in bits. The third and sixth columns indicate whether it is a register array. If it is a register array, the index value is shown.

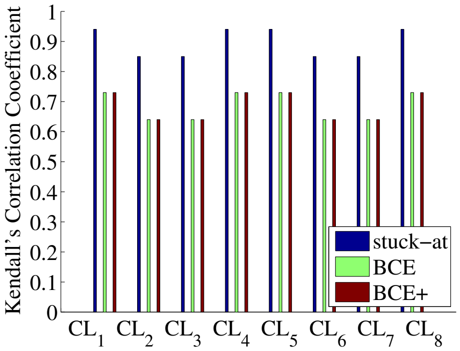

Figure 3. Correlation between output deviations and gate-level fault coverage (Parwan).

TABLE VI
WEIGHT VECTOR FOR SCHEDULER MODULE.

Registers	Fanin count	Fanout count	No. of connections	Weight
sb_counters	9211	4800	14011	0.0183
issued	112932	355464	468396	0.6113
valid	4886	355482	360368	0.4703
issue_head	2986	397086	400072	0.5221
issue_tail	146	202732	202878	0.2648
inst_array	275639	490584	766223	1
instout0	92962	4367	97329	0.1270
instout1	118629	4367	122996	0.1605
instout2	92553	4775	97328	0.1270
instout3	92638	4687	97325	0.1270
instout4	93045	4137	97182	0.1268
instout5	118337	4137	122474	0.1598

2) Experimental setup: The experimental setup is the same as that for Parwan processor, except for the calculation of deviations. For the Scheduler module, Design Compiler (DC) from Synopsys was used to extract the gate-level information for calculating weight vector. Synopsys Verilog Compiler (VCS) was used to run Verilog simulation and compute the deviations.

All experiments are based on 10 functional test sequences. Six of them are composed of instruction sequences, referred to as T_0, T_1, T_2, T_3, T_4, T_5. The other four, labeled as T_6, T_7, T_8 and T_9, are obtained indirectly by sequential ATPG. First, cycle-based gate-level stuck-at ATPG is carried out. From the generated patterns, stimuli are extracted and composed into a functional test sequence, labeled as T_6. Next we perform "not", "xnor", "xor" bit-operations on T_6 separately and obtain new functional test sequences T_7, T_8 and T_9. T_7 is obtained by inverting each bit of T_6; T_8 is obtained by performing "xnor" operation between the adjacent bits of T_6; T_9 is obtained by performing "xor" operation between the adjacent bits of T_6. The above bit-operations do not affect the clock and reset signals in the design.

3) RT-level deviations: Using the deviation-calculation method in Section III, we calculate the RT-level deviations for the Scheduler module by considering the parameter TC, weight vector and observability vector.

To obtain the weight vector, we first determine the fanin and fanout counts for each register or register array. A net is considered to be in the fanin cone of a register if there is a path through combinational logic from the net to that register. Similarly, a net is considered to be in the fanout cone of a register if there is a path through combinational logic from that register to the net. Table VI lists these counts for each register (array) and shows the weight vector components. In Column 4, "No. of connections" is the sum of the fanin count and the fanout count.

The observability vector is obtained using the dataflow diagram extracted from the design. The dataflow diagram is extracted in the same way as for the Parwan processor. Figure 4 shows the dataflow diagram of the Scheduler module.

We consider the following order of the registers: sb_counters, issued, valid, issue_head, issue_tail, instout0, instout1, instout2, instout3, instout4, instout5, and inst_array. The corresponding observability vector is $< 0.5,\ 0.5,\ 0.5,\ 0.3333,\ 1,\ 1,\ 1,\ 1,\ 1,\ 1,\ 0.5 >$.

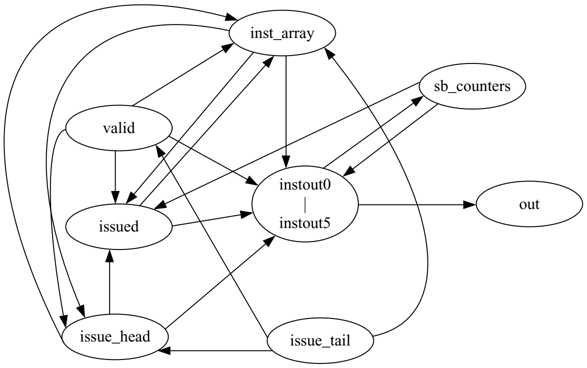

Figure 4. Dataflow diagram of Scheduler module.

TABLE VII
RT-LEVEL DEVIATIONS OF TEN FUNCTIONAL TESTS FOR SCHEDULER MODULE.

FT	Deviations							
	(CL_1)	(CL_2)	(CL_3)	(CL_4)	(CL_5)	(CL_6)	(CL_7)	(CL_8)
T_0	0.6250	0.4500	0.2817	0.2906	0.3628	0.5788	0.7704	0.2549
T_1	0.6261	0.4504	0.2828	0.2913	0.3630	0.5788	0.7714	0.2556
T_2	0.5986	0.4407	0.2698	0.2734	0.3586	0.5716	0.7456	0.2395
T_3	0.6054	0.4431	0.2732	0.2778	0.3597	0.5732	0.7522	0.2435
T_4	0.6195	0.4480	0.2794	0.2870	0.3619	0.5772	0.7653	0.2517
T_5	0.6257	0.4502	0.2821	0.2910	0.3629	0.5790	0.7710	0.2553
T_6	0.9773	0.7235	0.6991	0.7344	0.5119	0.7458	0.9966	0.6790
T_7	0.9815	0.7367	0.7132	0.7524	0.5211	0.7584	0.9975	0.6977
T_8	0.8556	0.5646	0.5192	0.4920	0.4180	0.6057	0.9451	0.4404
T_9	0.8826	0.5862	0.5551	0.5275	0.4293	0.6151	0.9598	0.4741

Given the information of the TCs, the weight vector, and the observability vector, RT-level deviations can be calculated for given CL vectors. Eight CL vectors were used for deviation calculation for the Scheduler module. They are listed as follows: CL_1 = (1, 0.99998, 0.99998, 1), CL_2 = (0.9999999, 0.999995, 0.999995, 0.9999999), CL_3 = (1, 0.99995, 0.99995, 0.9999998), CL_4 = (1, 0.999993, 0.999993, 1), CL_5 = (0.9999999, 0.999998, 0.999998, 0.9999999), CL_6 = (0.9999998, 0.999995, 0.999995, 1), CL_7 = (1, 0.99997, 0.99997, 1), and CL_8 = (1, 0.999994, 0.999994, 1). For CL_3 and CL_6, we set different values to the CL of 0→0 transition and 1→1 transition. The corresponding RT-level deviations under these eight CL vectors for the Scheduler module are shown in Table VII (FT stands for functional test). The total CPU time for deviation calculation and test-sequence grading is less than 8 hours. The CPU time for gate-level stuck-at (transition) fault simulation is 110 (175) hours, and the CPU time for computing the gate-level BCE (BCE^+) measure is 120 (125) hours, thus we are able to reduce CPU time significantly.

4) Correlation between output deviations and gate-level fault coverages: We obtain the stuck-at and transition fault coverages for the functional test sequences by running fault simulation at the gate-level. Bridging fault coverage is estimated using the BCE and BCE^+ metrics. The correlation between these gate-level fault coverage measures and RT-level deviations are computed and the Kendall's correlation coefficients are shown in Figure 5. As in the case of the Parwan processor, the correlation coefficients are close to 1 for all the CL vectors. This demonstrates that the RT-level deviations are a good predictor of the gate-level fault coverage.

5) Cumulative gate-level fault coverage (Ramp-up): We evaluate the cumulative stuck-at and transition fault coverage of several reordered functional test sequences.

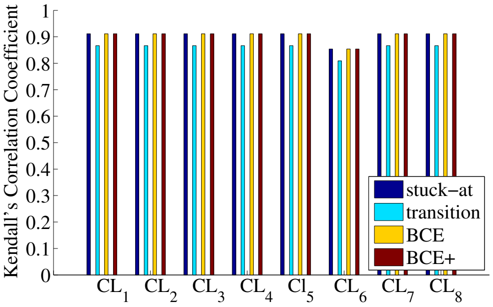

Figure 5. Correlation between output deviations and gate-level fault coverage (Scheduler module).

These reordered sequences sets are obtained in four ways: (i) baseline order $T_0, T_1 ..., T_9$; (ii) random ordering $T_2, T_1, T_5, T_7, T_0, T_3, T_8, T_9, T_6, T_4$; (iii) random ordering $T_4, T_1, T_0, T_9, T_2, T_7, T_6, T_3, T_5, T_8$; (iv) the descending order of output deviations. In the deviation-based method, test sequences with higher deviations are ranked first. Figure 6 and Figure 7 show cumulative stuck-at and transition fault coverages for the four reordered functional test sequences following the above four orders. We find that the deviation-based method results in the steepest cumulative stuck-at and transition coverage curves.

V. Conclusions

We have presented the output deviation metric at RT-level to model the quality of functional test sequences. This metric is based on the transition counts, observability vectors, and the weight vectors for registers. By adopting the deviation metric, timing-consuming fault simulation at gate-level can be avoided for the grading of functional test sequences. Experiments on the Parwan processor and the Scheduler module of the IVM design show that the deviations obtained at RT-level correlate well with the stuck-at, transition, and bridging fault coverage at the gate level. Moreover, the functional test sequences set reordered using deviations provides a steeper cumulative stuck-at and transition fault coverage, and there is an order of magnitude reduction in CPU time compared to gate-level methods.

References

[1] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, Boston; Kluwer Academic Publishers, 2000.

[2] P. C. Maxwell, I. Hartanto and L. Bentz, "Comparing Functional and Structural Tests," *in Proc. Int. Test Conf.*, pp. 400-407, 2000.

[3] A. K. Vij, "Good Scan= Good Quality Level? Well, It Depends...," *in Proc. Int. Test Conf.*, pp. 1195, 2002.

[4] J. Gatej et al., "Evaluating ATE Features in Terms of Test Escape Rates and Other Cost of Test Culprits," *in Proc. Int. Test Conf.*, pp. 1040-1049, 2002.

[5] J. Rearick and R. Rodgers, "Calibrating Clock Stretch During AC Scan Testing," *in Proc. Int. Test Conf.*, pp. 266-273, 2005.

[6] P. A. Thaker, V. D. Agrawal, and M. E. Zaghloul, "Register-transfer level fault modeling and test evaluationtechniques for VLSI circuits," *in Proc. Int. Test Conf.*, pp. 940-949, 2000.

[7] F. Corno, M. S. Reorda and G. Squillero, "RT-Level ITC'99 Benchmarks and First ATPG Result," *in IEEE Design & Test of Computers*, pp. 44-53, 2000.

[8] M. B. Santos, F. M. Goncalves, I. C. Teixeira, and J. P. Teixeira, "RTL-Based Functional Test Generation for High Defects Coverage in Digital Systems," *in Journal of Electronic Testing: Theory and Applications*, vol. 17, pp. 311-319, 2001.

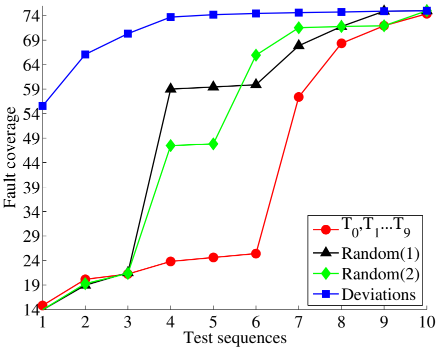

Figure 6. Cumulative stuck-at fault coverage for various test-sequence ordering methods (Scheduler module).

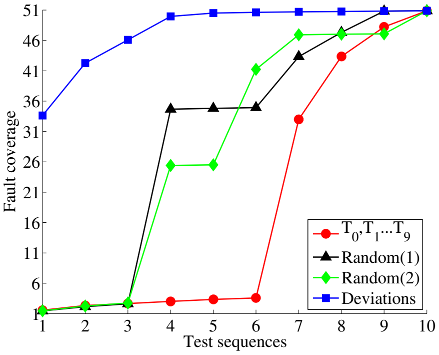

Figure 7. Cumulative transition fault coverage for various test-sequence ordering methods (Scheduler module).

[9] W. Mao and R. K. Gulati, "Improving Gate Level Fault Coverage by RTL Fault Grading," *in Proc. Int. Test Conf.*, pp. 150-159, 1996.

[10] S. Park et al.,"A Functional Coverage Metric for Estimating the Gate-level Fault Coverage of Functional Tests," *in Proc. Int. Test Conf.*, 2006.

[11] I. Pomeranz, P. K. Parvathala, S. Patil, "Estimating the Fault Coverage of Functional Test Sequences Without Fault Simulation," *in Proc. Asian Test Symposium*, pp. 25-32, 2007.

[12] T. E. Marchok et al., "A Complexity Analysis of Sequential ATPG," *IEEE Trans. CAD*, vol. 15, pp. 1409-1423, Nov. 1996.

[13] I. Pomeranz and S. M. Reddy, "LOCSTEP: A Logic-Simulation-Based Test Generation Procedure," *IEEE Trans. CAD*, vol. 16, pp. 544-554, May 1997.

[14] Z. Wang and K. Chakrabarty, "Test-Quality/Cost Optimization Using Output-Deviation-Based Reordering of Test Patterns," *IEEE Trans. CAD*, vol. 27, pp. 352-365, 2008.

[15] Z. Wang, K. Chakrabarty and M. Bienek, "A Seed-Selection Method to Increase Defect Coverage for LFSR-Reseeding-Based Test Compression," *in Proc. Europ. Test Symp.*, pp. 125-130, 2007.

[16] Z. Navabi, VHDL: *Analysis and Modeling of Digital Systems*, New York; McGraw-Hill Companies, 1997.

[17] N. J. Wang et al., "Characterizing the Effect of Transient Faults on a High-Performance Processor Pipeline," *in Proc. Int. Conf. Dep. Sys. and Net.*, pp. 61-70, 2004.

[18] N. J. Wang and S. J. Patel, "Restore: symptom based soft error detection in microprocessors," *in Proc. Int. Conf. Dep. Sys. and Net.*, pp. 30-39, 2005.

[19] M. Maniatakos et al., "Design and Evaluation of a Timestamp-Based Concurrent Error Detection Method (CED) in a Modern Microprocessor Controller," *in Proc. Int. Symp. DFT in VLSI Systems*, 2008.

[20] N. Karimi et al., "On the Correlation between Controller Faults and Instruction-Level Errors in Modern Microprocessors," *in Proc. Int. Test Conf.*, 2008.

[21] B. J. Chalmers, *Understanding Statistics*; CRC Press, 1987.

[22] Documentation for the Parwan processor, http: //mesdat.ucsd.edu/ ∼lichen/260c/parwan/.

[23] B. Benware et al., "Impact of Multiple-Detect Test Patterns on Product Quality," *in Proc. International Test Conference*, pp. 1031-1040, 2003.

[24] H. Tang et al., "Defect Aware Test Patterns," *in Proc. Design, Automation, and Test in Europe Conf.*, pp. 450-455, 2005.