Performance Evaluation of Software-based Error Detection Mechanisms for Localizing Electrical Timing Failures under Dynamic Supply Noise

Yutaka Masuda Masanori Hashimoto Takao Onoye
Department of Information Systems Engineering, Osaka University
Email: {masuda.yutaka, hasimoto}@ist.osaka-u.ac.jp, TEL: 06-6879-4528, FAX: 06-6879-4529

***Abstract*—For facilitating error localization, software-based error detection techniques have been proposed and EDM (error detection mechanisms) transformation is one of these techniques. To discuss the effectiveness of EDM for electrical bug localization, two scenarios are considered; (1) localizing an electrical bug occurred in the original program, and (2) localizing as many potential bugs as possible. We experimentally evaluated the error detection performance in these two scenarios under dynamic power supply noise. Experimental results show that the EDM transformation customized for quick error detection cannot locate electrical bugs in the original program in the firs scenario, but it is useful for findin potential bugs in the second scenario.**

***Index Terms*—electrical bug, software-based error detection, EDM transformation, error detection**

I. Introduction

Electrical timing bug, which causes a timing failure in a logically correct design due to electrical property of the chip, is becoming one of the most serious concerns in post-silicon validation. Electrical timing bugs originate from supply voltage variation, temperature gradient, crosstalk noise, and so on [1], and these factors dynamically fluctuat depending on the circuit operation, such as a program running on a processor, and operation environment. In design time, accurately predicting the occurrence of electrical timing bugs and their condition is difficult and hence unexpected electrical timing bugs are often observed in post-silicon validation.

Post-silicon validation gives a wide variety of test patterns at various operating conditions. Once an unexpected system behavior is observed, we start on analyzing the circuit operation. In this analysis, we need to (1) notice error occurrence, (2) localize the error in place, e.g. ALU and cache controller, and time, and (3) manifest the occurrence condition [1]. The most efforts for this analysis are made in (1) and (2) [2], and hence reducing these efforts is highly demanded.

Error occurrence is often detected by observing abnormal behaviors, such as system crash, segmentation fault, and invalid op code. End-result-check, which compares the execution result with the expected value, can be also used to fin error occurrence. The next step is error localization and it is challenging, because the time interval between the error occurrence and the detection of such an abnormal behavior is quite long. It sometimes reaches billions of cycles [3]. Due to such a long error detection latency, it is difficul to know when and where it occurred, since the trace buffer, which is often used to record signals on a chip for post-silicon debug, has limited record depth of, for example, thousands cycles [4]. Therefore, reducing the error detection latency is helpful to facilitate the error localization, and for this purpose, Park et al. proposed IFRA (Instruction Footprint Recording and Analysis) that detected error quickly by capturing error indication and analyzed the operational record using added hardware [4]. Assertion-based error detection with additional hardware is also proposed [5], [6]. In this approach, it is important when, where and how assertions are inserted for efficien detection with smaller area overhead [7].

Another approach that reduces error detection latency is software based approach, and QED (Quick Error Detection) transformation [3] is one of the software based methods. QED decomposes the input program into blocks, and duplicates each block within the program at assembly level. Also for every pair of the original and duplicated blocks, QED inserts a register-level consistency check that compares calculation results. With this fine-graine checking, QED succeeded in dramatically reducing error detection latency. Reference [3] reported that for specifi logic bugs, QED improved error detection latency by six orders of magnitude, i.e. from billions of cycles to a few thousand cycles. This shorter error detection latency helps improve the efficien y of post-silicon validation.

EDM (Error Detection Mechanisms) transformation [8] is another software-based error detection approach which was originally proposed for detecting soft errors. Reference [8] reported that for random bit flip injected to data memory the error detection coverage was over 90%. In [9], the coverage of over 80% was achieved for a single bit fli occurred in registers. EDM adds data and code redundancy to an input program written in a high-level source code (e.g. C and C++), and generates a special program. Here, various programs, e.g., random instruction tests, architecture-specifi focused tests, and end-user applications such as operating systems and games, can be given as an input program. The main advantage of EDM transformation lies in the fact that it can be applied to a high-level source code independent of the underlying hardware. To detect errors affecting data, EDM duplicates each variable in the program and adds consistency checks after every read operation. Here, the consistency check after the read operation makes the error detection latency longer while it is acceptable for soft error detection. Therefore, we devise another EDM implementation that adds consistency checks after every write operation aiming at shorter error detection latency. We will evaluate both EDM implementations in this paper.

On the other hand, the performance of software-based error detection techniques for electrical bug has not been studied explicitly, and their effectiveness against electrical bugs is not clear. This paper focuses on the performance of EDM for electrical timing error localization, and presents a case

978-1-4673-8388-2/15/$31.00 ©2015 IEEE

study that considers program-dependent supply noise with a supply noise aware simulation framework supposing supply noise is the most primary source of electrical bugs. Here, we consider two scenarios of EDM usage in post-silicon validation; (1) localizing the exact electrical bug occurred in the original program, and (2) localizing as many potential errors as possible which could lead to abnormal behaviors. For the firs scenario, two necessary conditions must be satisfied error reproducibility, and diversity between the executions of the duplicated blocks. On the other hand, for the second scenario, only the diversity must be satisfied We discuss the utility of EDM for electrical bug localization in these two scenarios on the basis of experimental results.

The rest of this paper is organized as follows. Section II explains EDM transformations and examines the necessary conditions in which EDM localizes an electrical bug. Section III presents a case study that investigates whether EDM transformation is helpful to localize an electrical bugs due to dynamic supply noise. Section IV examines the experimental results. Lastly, concluding remarks are given in Section V.

II. Localizing electrical bug with EDM

This section explains EDM transformations, discusses two scenarios of EDM usage in post-silicon validation, and describes the necessary conditions for error localization in each scenario.

A. EDM transformation

To detect an error quickly after its occurrence, EDM converts an input program to a special program using several transformation techniques. The EDM transformation and error detection described in [8] are exemplifie in Fig. 1, where the original EDM transformation in [8] is hereafter called EDM-O. First, EDM-O divides an input program into blocks and duplicates each block, where each block size is equal to the interval between variable read operations. The paired original and duplicated blocks are aligned in sequence. Second, for all the pairs of the original and duplicated blocks, EDM-O inserts check instructions to compare the execution results. Consequently, the EDM-O-transformed program executes the original block, the duplicated block and the check instruction in sequence for all the pairs of the original and duplicated blocks. This transformation is performed at C/C++ level. Fig. 2 gives an example of EDM-O-transformed code. Here, EDM-O is developed for the purpose of improving soft-error detection coverage [8], and hence the check is constantly inserted after each variable read to know whether bit flip occurred in the memory, registers, or FFs.

EDM-O is useful for soft error detection, but it can be improved for shortening error detection latency. The left figur of Fig. 3 illustrates such an example. Suppose the memory write of variable $a0$ in the firs line failed. In this case, the memory of $a0$ is not accessed for a long time and the inserted check is performed after a long time passes. To shorten the error latency, the check should be performed immediately after the memory write access (right figur of Fig. 3). Motivated by this,

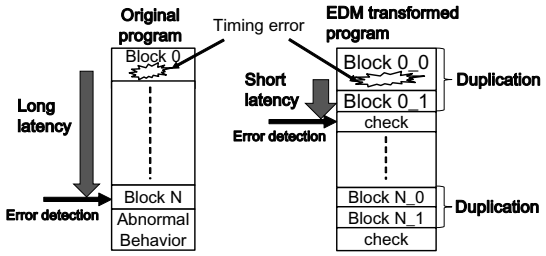

Fig. 1. Error detection by EDM transformation.

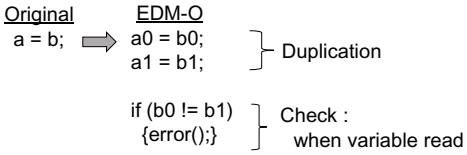

Fig. 2. An example of EDM-O code.

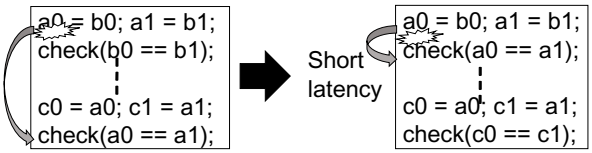

Fig. 3. Difference of error detection latency between EDM-O and EDM-L.

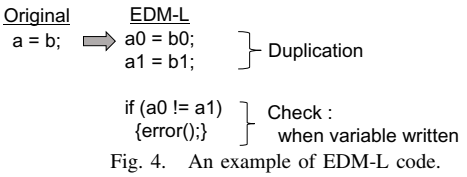

Fig. 4. An example of EDM-L code.

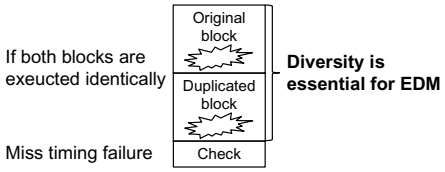

Fig. 5. Diversity is necessary for error detection.

we devised EDM-L (EDM for short Latency). Fig. 4 shows an example of EDM-L-transformed code. EDM-L inserts check instructions for every variable write. Consequently, when an error occurs in the original block, we can expect that the next check instruction detects the error occurrence.

Furthermore, for detecting the error occurrence with the check instruction, the diversity between the original block and the duplicated block is crucially important. If the original block and the duplicated block are identical, the same error would occur in both the blocks and the check instruction fails to detect the error as illustrated in Fig. 5. In the EDM transformation, the original blocks and the duplicated blocks often split the memory space to gain the diversity, where the different memory addresses are expected to have different access times. EDM can include various transformations to maximize the diversity.

B. EDM usage scenarios and necessary conditions for error detection

In this section, we list two EDM usage scenarios in post-silicon validation, and discuss the necessary conditions that EDM needs to satisfy in each scenario.

We consider the following two scenarios.

Scenario1:
> When an original program was running, an electrical bug occurred. We want to localize this error using EDM transformation.

Scenario2:
> We want to localize as many potential bugs as possible.

We firs examine the necessary conditions for the firs scenario. In Scenario1, EDM should satisfy the two conditions below simultaneously (Fig. 6).

COND1:
> EDM-transformed program reproduces the error which occurred in the original input program.

COND2:
> EDM induces enough diversity so that the paired original and duplicated blocks output different computational values.

The firs COND1 condition is necessary to investigate the root cause of the error observed in the original program. To reproduce the error occurrence, the EDM-transformed program should maintain the similar behavior of the original program. If EDM does not reproduce the same error, the error localization of the original program is impossible.

The second COND2 is the fundamental condition for EDM to work. If the original and duplicated blocks output the same wrong values, the inserted check instruction misses the error. Focusing on the second COND2 condition, dynamically fluctuatin factors, such as supply noise, might help increase the diversity. The diversity originates from the timing characteristics of the fabricated chip under test and dynamically fluctuatin factors. For example, power supply noise changes depending on the running program, which may improve the diversity.

On the other hand, COND1 is thought to become more difficul to satisfy as dynamically fluctuatin factors become more significant The supply noise, for example, of the chip on which the original program is running can be different from the noise of the EDM-transformed program. In this case, the error observed in the original program may disappear in the EDM-transformed program, and a new error may arise at another program location.

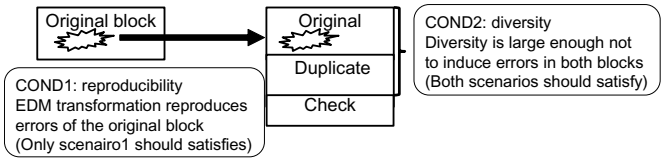

Fig. 6. Two conditions for EDM to localize electrical bug in Scenario1.

As stated above, Scenario1 requires that both COND1 and COND2 are satisfied However, previous studies did not focus on COND1. It is not clear whether or how often COND1 can be satisfie in the EDM-transformed programs. In addition, it is not clear whether EDM satisfie COND2 for realistic electrical bug. The next section experimentally investigates whether these two conditions are satisfie under dynamic power supply noise.

In Scenario2, the error observed in the original program does not need to be reproduced. Moreover, inducing a new error could be preferable, since potential bugs could be localized. Therefore, COND1 is not necessary. Only COND2 needs to be satisfied The necessary condition for Scenario2 is the subset of the condition for Scenario1 and hence the experiments for Scenario1 are valid for Scenario2 as well.

III. Experimental Evaluation of EDM Transformation

This section experimentally investigates whether EDM transformation works well in Scenario1 and Scenario2. This experiment supposes that dynamic power supply noise is the primary source of electrical timing bugs, and accurately reproduces the impact of EDM on the dynamic supply noise and the consequent timing variations.

A. Experimental Setup

Our experimental evaluation was performed for an industrial embedded processor (Toshiba MeP processor). This processor was synthesized and laid out with an industrial 65nm library. In this experiment, the post-layout design was used for the simulations which will be explained later. We took three C-language benchmark programs, dijkstra, crc, and sha from MiBench [10] as original input programs. We implemented an EDM translator and used this translator to get EDM-transformed programs.

In EDM transformation, two types of check instructions are inserted [8]; (1) data check and (2) code fl w check. For the data checking, each variable v is duplicated as $v0$ and $v1$. Then, the consistency check is performed every time $v0$ ($v1$) is read in EDM-O. In EDM-L, the data check is performed every time $v0$ ($v1$) is written. The code fl w check aims to detect an illegal change of the code execution fl w, such as an illegal jump operation. The code fl w check is inserted as follows.

First, EDM identifie all the basic blocks, i.e., branch-free sequences, in the program and checks whether all the statements in every basic block are executed in sequence by numbering the basic blocks. Second, checks for every test statement (e.g. if, else if, while) are inserted. EDM inserts the opposite test to both the true and false clauses to detect an illegal execution fl w. The last target is call and return instructions. Every procedure, i.e., function in the program, is associated with its unique number, and the number is checked for every call of the procedure. In this section, we duplicated all variables, and inserted all data checks and code fl w checks.

TABLE I
IMPACT OF EDM-L TRANSFORMATION ON CYCLE TIME AND CACHE MISS.

	execution cycles		inst. cache misses		data cache misses	
	orig.	EDM	orig.	EDM	orig.	EDM
dijkstra	24512 (1.00)	69838 (2.85)	45 (1.00)	161 (3.58)	11 (1.00)	20 (1.82)
sha	30757 (1.00)	97831 (3.18)	44 (1.00)	167 (3.80)	25 (1.00)	42 (1.68)
crc	19975 (1.00)	57252 (2.87)	9 (1.00)	29 (3.22)	35 (1.00)	71 (2.03)

A value in parentheses is the ratio of full-EDM-L divided by original.

TABLE II
IMPACT OF EDM-O TRANSFORMATION ON CYCLE TIME AND CACHE MISS.

	execution cycles		inst. cache misses		data cache misses	
	orig.	EDM	orig.	EDM	orig.	EDM
dijkstra	24512 (1.00)	65693 (2.68)	45 (1.00)	150 (3.33)	11 (1.00)	20 (1.82)
sha	30757 (1.00)	120487 (3.92)	44 (1.00)	213 (4.84)	25 (1.00)	52 (2.08)
crc	19975 (1.00)	60000 (3.00)	9 (1.00)	23 (2.56)	35 (1.00)	65 (1.86)

A value in parentheses is the ratio of full-EDM-O divided by original.

We call this transformation as full-EDM. Sparser duplication and data check will be discussed in the next section.

For the original and duplicated blocks, the same input data was stored at two different addresses of data memory, and each block accessed its own data in the data memory. Table I and II list increases in the number of execution cycles and the number of cache misses by full-EDM-L and full-EDM-O, respectively. The number of execution cycles increased three to four times, and the increase in the number of instruction cache misses was similar. The increase in the data cache miss was roughly double, which is consistent with a fact that the data size is doubled in the full-EDM-transformed program.

We evaluated and compared the error occurrences in the original program and EDM transformed programs by logic simulation. Our logic simulation framework concurrently simulates two MeP designs; one is at RT (register transfer) level and the other is at gate level. The RT-level logic simulation is performed with zero-delay model, and hence the output is always correct disregarding the clock frequency and the given supply voltage. On the other hand, the gate-level logic simulation includes timing information and then may output wrong values. In this work, a noise-aware logic simulation method [11] is adopted to take explicitly into consideration program-dependent dynamic supply noise. This simulation method will be explained in the next subsection. Once an inconsistency is detected at a FF between RT-level and gate-level simulations, we can immediately notice a timing error occurrence. Thanks to this, we can know the exact location of timing error in time and space.

The comparisons of error occurrence between the original and full-EDM-L programs were performed for the following 300 situations. Due to manufacturing variability, each chip has different delay characteristics. To reproduce this, we virtually fabricated 10 MeP chips assuming that each instance delay randomly fluctuate with the standard deviation of 25% of the typical instance delay. In addition, depending on the fina products, the LSI package may change. We assumed 10 package conditions, i.e. 10 sets of equivalent circuit parameters of power distribution network. The equivalent circuit model will be shown in the next subsection. In summary, 100 = 10 $\times$ 10 samples were evaluated for each program, i.e. 300 samples in total. Similarly, the full-EDM-O program was evaluated in 300 situations.

We focused on the firs error occurred in the program execution, and its location was considered to check whether COND1 was satisfied The minimum clock cycle that caused timing errors was searched with 2ps interval. When we decomposed the program into blocks, we numbered the blocks from the beginning. We regarded the difference of the block numbers as the proximity of error occurrence locations. When the difference is zero, the timing error is reproduced at the same block in the EDM-transformed program and COND1 is satisfied COND2 was evaluated by checking whether the program was terminated by the check instructions. Even if a timing error occurred in the EDM-transformed programs, the check instructions sometimes miss the error. This case can be categorized into two groups; silent error and masked error. In the silent error case, the execution result is different from the correct result. On the other hand, in the case of masked error, the execution result is correct.

B. Noise-aware logic simulation

In this paper, we used a noise-aware logic simulation method that could consider dynamic power supply noise in gate-level logic simulation [11]. The dependence of gate delay on supply voltage was evaluated with HSPICE and expressed using a delay element whose delay is controlled by digital signals representing the supply voltage. Here, this delay element is described at RTL. By attaching this delay element to every gate, we can reproduce voltage-dependent gate delay. In addition, we can dynamically vary the gate delay by changing the digital signal that represents supply voltage.

When performing the above noise-aware logic simulation, we need to give a waveform of dynamic power supply noise. We prepared noise waveform information with the following two steps. First, we simulated the post-layout MeP design with the original and EDM transformed programs by a transistor-level circuit simulator, and obtained waveforms of the current consumed by MeP for each program. Here, it should be noted that the transistor-level simulation to obtain the current waveform is very time consuming and it took three days for sha-full-EDM-O program. To make the CPU time needed for the entire evaluation in this work acceptable, the two-step procedure was adopted. Next, we gave this current waveform to the equivalent circuit of Fig. 7 and obtained the waveform of dynamic power supply noise. The nominal supply voltage was 1.0V. To reproduce various package assemblies and obtain corresponding noise waveforms, we used 10 sets of PDN parameters in Fig. 7. The parameter setting is explained in the following.

We varied three parameters of C_{PKG}, $R_{\mathrm{ESR\_PKG}}$ and $L_{\mathrm{ESL\_PKG}}$ representing the package capacitor, and one

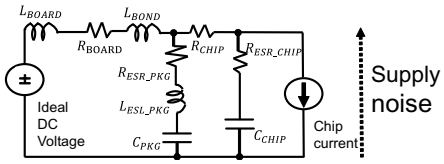

Fig. 7. An equivalent circuit of power distribution network.

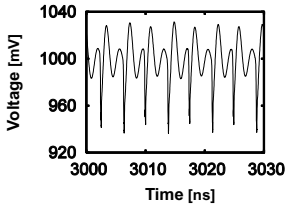

Fig. 8. A waveform example of inductive fluctuation

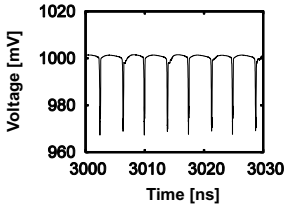

Fig. 9. A waveform example of resistive drop.

parameter of C_{CHIP} representing the on-chip capacitor. The other five parameters were fixed as follows; L_{BOARD}=0.1nH, R_{BOARD}=5mΩ, L_{BOND}=0.3nH, R_{CHIP}=0.1Ω and $R_{\mathrm{ESR\_CHIP}}$=0.3Ω. We prepared five configuration of the package capacitor; (1) no package capacitor, (2) one NPO capacitor, (3) one X7R capacitor, (4) ten NPO capacitors in parallel and (5) ten X7R capacitors in parallel, where NPO and X7R are commercially available popular ceramic capacitors [12]. C_{PKG}, $R_{\mathrm{ESR\_PKG}}$ and $L_{\mathrm{ESL\_PKG}}$ of NPO and X7R are (100pF, 0.3Ω, 0.6nH) and (1nF, 0.6Ω, 0.6nH), respectively [12], [13]. As for the on-chip capacitance C_{CHIP}, two values of 3.5nF and 0.3nF were prepared. Consequently, 10 (=5×2) sets of PDN parameters were prepared and used to obtain the noise waveforms. Examples of the noise waveforms are shown in Figs. 8 and 9. These noise waveforms were given to the noise-aware logic simulation.

C. Evaluation results

Fig. 10 shows how many samples satisfied COND1 of reproducibility and COND2 of diversity. For every timing error in the original program, we checked if COND1 and COND2 are satisfied in the EDM-transformed program. Among 600 timing error samples, we could not find a sample that satisfied COND1 and COND2 simultaneously, which suggests EDM is less helpful in Scenario1. In addition, over 75% of errors satisfied neither COND1 nor COND2. Comparing full-EDM-L with full-EDM-O, we can see the difference in the proportion that only COND1/COND2 is satisfied In the following, we examine the results for COND1 and COND2 separately in detail.

1) COND1: Figs. 11 and 12 show the proximity of the errors occurred in the original and full-EDM dijkstra, crc, and sha programs. Remind that the proximity is defined as the difference of the block numbers of the error occurrence. The block number difference of zero means that the same error is observed in the original and EDM-transformed programs. In EDM-L, 10% of errors in crc and 2% of errors in dijkstra were reproduced. On the other hand, in sha, no errors were reproduced. In EDM-O, over 30% of errors were reproduced in crc, but no errors were reproduced in dijkstra and sha. As

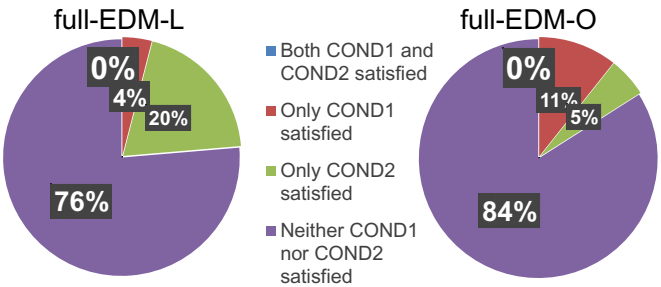

Fig. 10. Evaluation results of full-EDM.

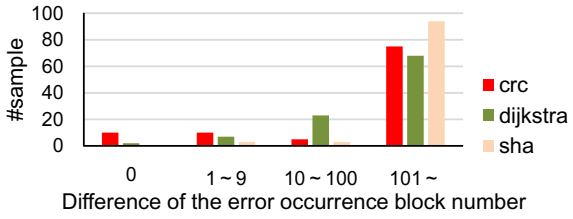

Fig. 11. COND1: difference of block numbers of first error occurrence between original and full-EDM-L programs. For each program, the number of samples is 100.

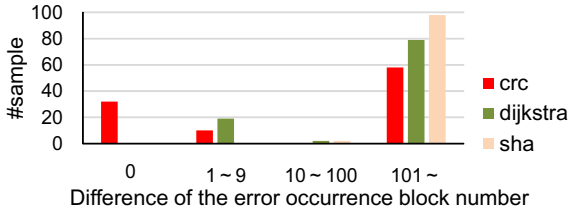

Fig. 12. COND1: difference of block numbers of first error occurrence between original and full-EDM-O programs. For each program, the number of samples is 100.

a whole, EDM-L and EDM-O reproduced only 4% and 11% errors, respectively. Such low reproduction ratios are mainly due to the following two reasons.

The first reason is that EDM changes supply voltage noise since the block duplication and check insertion change the instruction sequence and the usage of circuit blocks, such as memory and general purpose registers. In other words, even when the same instructions are performed, the supply noise could change, because the used registers and memory addresses are different and the inductive noises excited in the previous clock cycles are superposed. Fig. 14 shows a comparison of noise waveforms between the original and full-EDM dijkstra programs, where the same instruction was performed in this clock cycle. We can see that the voltage waveforms are not identical. For further investigation, we evaluated the minimum supply voltage within a clock cycle every time mov instruction was performed. Fig. 13 shows a histogram of the minimum voltage in the original crc program. We can see that the minimum voltage value ranges from 941 mV to 947 mV even though the same instruction of mov is performed. This waveform difference prevents the error reproduction.

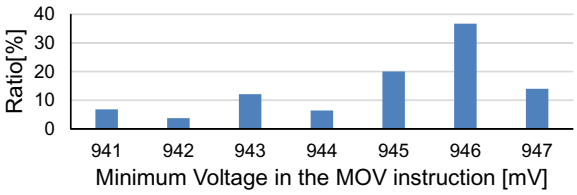

Fig. 13. The histogram of the supply voltage when MOV instructions were executed (crc-original).

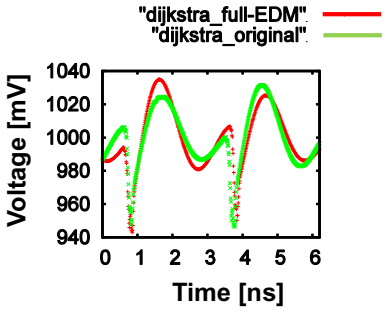

Fig. 14. Voltage inconsistency between original and full-EDM programs.

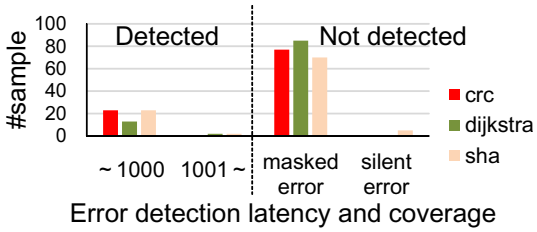

Fig. 16. COND2 : Error classificatio in full-EDM-L. For each program, the number of samples is 100.

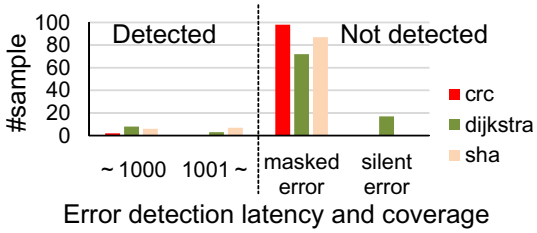

Fig. 17. COND2 : Error classificatio in full-EDM-O. For each program, the number of samples is 100.

The second reason is that EDM lengthens the program execution as previously shown in Tables I and II. As the program becomes longer, a new timing error, which is different from the error observed in the original program, is more likely to occur. In addition, the duplication and frequent check insertion change the instruction composition of the program. Fig. 15 shows the ratios of the instructions executed in sha-original and sha-full-EDM-L programs, respectively. We can see that instruction ratios of the EDM and original programs are considerably different. For example, in EDM, the number of lw (load word) instruction executions increases because the used memory space is doubled, and a number of beq (branch if equal) instructions are introduced due to check insertion. These instruction variations not only affect the processor behavior but also enlarge the noise difference, which makes the error reproduction difficult

2) COND2: Next, COND2 is examined. Figs. 16 and 17 show the proportions of silent errors, masked errors and detected errors. For detected errors, the histogram of the error detection latency is presented. In EDM-L, we can see that 77% of errors are masked and 2% are silent errors, whereas 87% are masked errors and 7% are silent errors in EDM-O.

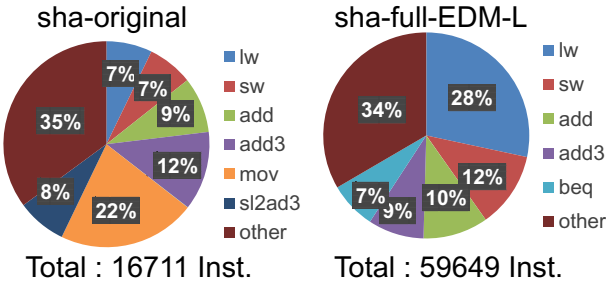

Fig. 15. Proportion of executed instructions in sha-original, sha-full-EDM-L.

In other words, most of the electrical bugs did not propagate to the memory and general purpose registers.

Among the non-masked errors, 86% errors were detected within 1000 cycles in EDM-L, while 30% in EDM-O. This result indicates that the EDM-L performance of detecting electrical bugs that affect execution results is high. We can say EDM-L is helpful to detect noise induced errors with shorter error detection latency. In other words, we can use EDM-L in Scenario2. For the errors having long error detection latency, we found a tendency that the firs error was not detected and the second or later error was detected by the check instruction. On the other hand, the EDM-O performance was not good. The proportion of silent errors was larger, and the detection latency was longer. Clearly, for the purpose of quick error detection in post-silicon validation, EDM-L is much better than EDM-O.

IV. Discussion

The experimental results in the previous section pointed out that both EDM-O and EDM-L transformations were not effective in Scenario1 whereas EDM-L transformation was effective in Scenario2. The problem was that COND1 was not satisfie in most cases since EDM transformation considerably changed the program. This observation could lead us to investigate whether the proportion of COND1 satisfaction could be improved by reducing the amount of program modification

A key factor that controls the amount of program modifica tion is the frequency of check insertion, and it is expected to affect the reproducibility, detectability and detection latency. For example, more frequent check insertion often contributes to fewer silent errors and shorter error detection latency. On the other hand, as the check insertion becomes less frequent, longer instruction sequences, which are similar with those in the original program, consequently similar supply noises are expected to arise. This trade-off is considered with two parameters of inst_max and inst_min in QED [3].

TABLE III
NUMBER OF INSERTED CHECK INSTRUCTIONS IN EDM-L.

	full-EDM	less-data-check-EDM	no-code-check-EDM
dijkstra	1950 (1.00)	1237 (0.63)	1061 (0.54)
sha	5015 (1.00)	2450 (0.49)	2939 (0.59)
crc	3017 (1.00)	2145 (0.71)	1637 (0.54)

A value in parentheses is the ratio divided by full-EDM-L.

In this section, we evaluate the impact of the frequency of check insertion on the EDM-L performance both in Scenario1 and Scenario2. For this purpose, we generated two EDM transformed programs for dijkstra, crc and sha programs as follows.

- less-data-check-EDM : For some of variables which were randomly selected, we duplicated and inserted data checks. In dijkstra, 3 out of 17 variables, in crc, 2 out of 15 variables, in sha, 2 out of 10 variables were duplicated and checked respectively. All the checks for code fl w were inserted.
- no-code-check-EDM : We duplicated all the variable and all the data checks. No check instructions for code fl w were inserted.

Here, full-EDM, which was used in the previous section, inserted checks the most frequently, and here we reduced check insertion from full-EDM. Table III lists the decrease in the number of inserted check instructions in EDM programs. The check instructions were reduced by 51 to 29 % in less-data-check-EDM and by 41 to 46% in no-code-check-EDM. From Tables I and III, we can also see that check instructions were executed in full-EDM per 15 cycles to 35 cycles in average.

A. Scenario1

Fig. 18 shows the evaluation results for less-data-check-EDM-L and no-code-check-EDM-L. Similarly to the full-EDM evaluation, 300 error samples were evaluated for each EDM-L transformation. Again, there are few cases that satisfie both COND1 and COND2, and the ratios for less-data-check-EDM-L and no-code-check-EDM-L were 0% and 2%, respectively.

Compared to Fig. 10, less-data-check-EDM-L improved the COND1 satisfaction ratio slightly from 4% to 6%. Fig. 19 shows the instruction constitution of the sha-no-code-check-EDM-L. Compared to Fig. 15, the instruction constitution of the sha-no-code-check-EDM-L becomes closer to that of sha-original as we expected. Table IV lists the statistics of the supply noise. No-code-check-EDM-L might improve the similarity, but this tendency cannot be clearly extracted from these statistics. On the other hand, the COND2 satisfaction of less-data-check-EDM-L degraded from 20% to 12%.

Summarizing the results, there is a trade-off between the COND1 satisfaction and COND2 satisfaction in terms of the check insertion frequency, and it is difficul to improve COND1 and COND2 satisfactions simultaneously. Furthermore, less frequent check insertion improves the COND1 satisfaction little, while it degrades the COND2 satisfaction significantl . From this tendency, we conclude it is difficul to fin the appropriate frequency of check insertion in this trade-off for Scenario1.

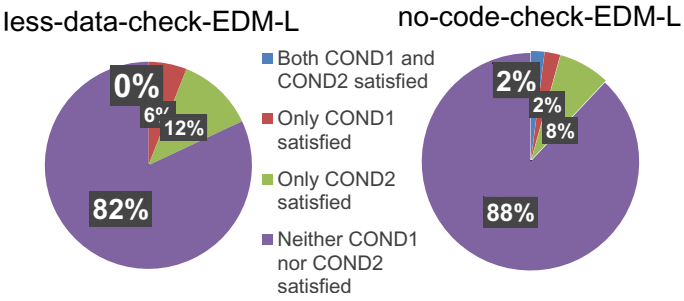

Fig. 18. Evaluation results of less-data-check and no-code-check-EDM-L.

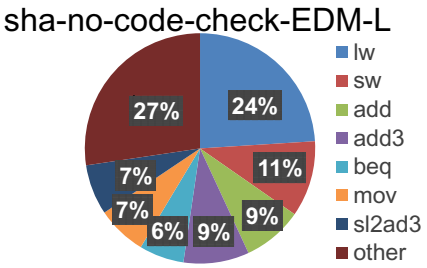

Fig. 19. Proportion of executed instructions in sha-no-code-check-EDM-L.

TABLE IV
STATISTICS OF SUPPLY VOLTAGE NOISES.

	Average	Minimum	Maximum	Standard Deviation
original	999.5 mV	954.5 mV	1013.6 mV	9.2 mV
full-EDM-L	999.0 mV	954.9 mV	1013.5 mV	10.6 mV
no-code-check -EDM-L	999.2 mV	954.5 mV	1013.7mV	10.5 mV

B. Scenario2

Figs. 20 and 21 show the proportions of silent errors, masked errors and detected errors in less-data-check-EDM-L and no-code-check-EDM-L. Compared to Fig. 16, the error detection proportion degraded from 21% to 14% in less-data-check-EDM-L and from 21% to 10% in no-code-check-EDM-L. However, in no-code-check-EDM-L, the detectability for the non-masked errors is quite high and it was 96%. This result suggests that no-code-check-EDM-L is also helpful to localize electrical bugs in Scenario2.

If full-EDM-L and no-code-check-EDM-L could detect different potential bugs, using both the EDM transformations would improve the quality of post-silicon validation. We compared the errors detected by these two EDM transformations. Fig. 22 shows the relation between the block number of error occurrence in dijkstra-full-EDM-L and that in dijkstra-no-code-check-EDM-L. The number of dots corresponds to the number of detected errors. We can see that the dots spread out. This means that these two EDM transformations detected the errors occurred in different blocks, i.e. different errors. Fig. 23 shows the instructions that caused errors in sha-full-EDM-L and sha-no-code-check-EDM-L. The proportions of

Fig. 20. COND2 : Error classificatio in less-data-check-EDM-L. For each program, the number of samples is 100.

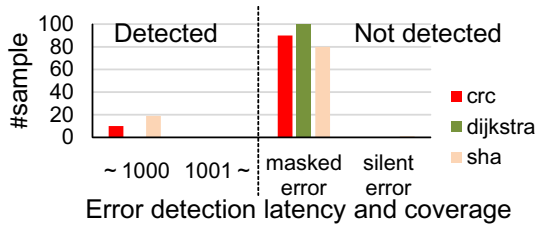

Fig. 21. COND2 : Error classificatio in no-code-check-EDM-L. For each program, the number of samples is 100.

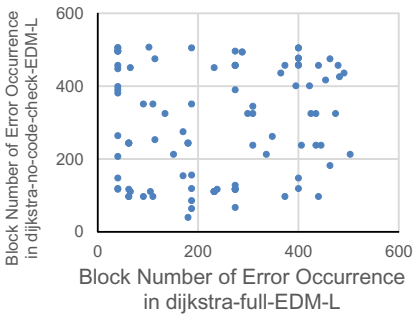

Fig. 22. Block numbers of error occurrence in dijkstra-full-EDM-L and dijkstra-no-code-check-EDM-L.

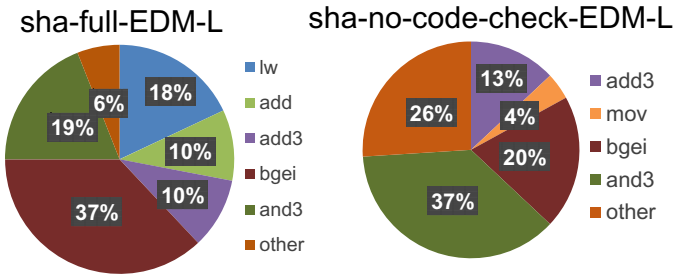

Fig. 23. Error occurrence instructions in sha-full-EDM-L and sha-no-code-check-EDM-L.

instructions that caused errors are much different. These results clarify that these two EDM-L programs behave distinctly and detect different potential bugs. By changing the check insertion frequency, we might be able to generate various programs for post-silicon validation and improve the capability of potential bugs localization.

V. Conclusion

This work experimentally evaluated the error detection performance of the EDM transformation, which is one of C-level code transformations, for supply noise induced timing errors. To discuss the effectiveness of the EDM for electrical bug localization, we consider two EDM usage scenarios; localizing an electrical bug occurred in the original program (Scenario1), and localizing as many potential bugs as possible (Scenario2). For these two scenarios, we evaluated the error localization performance of EDM-O and EDM-L transformations under dynamic power supply noise, where EDM-O is the original EDM reported in literature and EDM-L is the EDM modifie for shorter error detection latency. Experimental results show that both EDM-L and EDM-O cannot locate the same electrical bugs observed in the original program in Scenario1 since the error reproduction is prevented by noise waveform alternation and longer program execution originating from EDM transformation. On the other hand, in Scenario2, EDM-L is useful for findin potential bugs because the EDM-L performance of detecting electrical bugs affecting execution results is high. Also, generating various EDM transformed programs with different check insertion frequencies could help improve the quality of post-silicon validation.

Acknowledgement

This work is partly supported by STARC.

References

[1] P. Patra, "On the cusp of a validation wall,"*Design & Test of Computers*, vol. 24, no. 2, pp.193–196, June 2007.
[2] D. Josephson, "The good, the bad, and the ugly of silicon debug," *Proc. DAC*, pp.3–6, 2006.
[3] D. Lin, T. Hong, Y. Li, S. Eswaran, S. Kumar, F. Fallah, N. Hakim, D.-S. Gardner, and S. Mitra, "Effective Post-Silicon Validation of System-on-Chips Using Quick Error Detection," *IEEE Trans. CAD* , vol. 33, no. 10, pp.1573–1590, Oct. 2014
[4] S.-B. Park, T. Hong, and S. Mitra, "Post-silicon bug localization in processors using instruction footprint recording and analysis (IFRA)," *IEEE Trans. CAD*, vol. 28, no. 10, pp.1545–1558, Oct. 2009.
[5] A.A. Bayazit, S. Malik, "Complementary use of runtime validation and model checking," *Proc. ICCAD*, pp.1052–1059, 2005.
[6] M. Boule, Z. Zilic, "Incorporating efficien assertion checkers into hardware emulation," *Proc. ICCD*, pp.221–228, 2005.
[7] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," *Proc. DAC*, pp.12–17, 2010.
[8] M. Rebaudengo, M.S. Reorda, M. Torchiano, and M. Violante, "Soft-error detection through software fault-tolerance techniques," *Proc. DFT*, pp.210–218, 1999.
[9] N. Nicolescu, R. Velazco, "Detecting soft errors by a purely software approach: method, tools and experimental results," *Proc. DATE*, pp.57–62, 2003.
[10] M.R. Guthaus, J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge, and R.B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," *Proc. Workload Characterization*, pp.3–14, 2001.
[11] M. Ueno, M. Hashimoto, and T. Onoye, "Real-time On-chip Supply Voltage Sensor and Its Applicatoin to Trace-based Timing Error Localization," *Proc. IOLTS*, pp.188–193, 2015.
[12] L.D. Smith, R.E. Anderson, D.W. Forehand, T.J. Pelc, and T. Roy, "Power distribution system design methodology and capacitor selection for modern CMOS technology," *IEEE Trans. Advanced Packaging*, vol.22, no.3, pp.284–291, Aug 1999.
[13] T. Roy,L. Smith, and J. Prymak, "ESR and ESL of ceramic capacitor applied to decoupling applications," *Proc. EPEP*, pp.213–216, 1998.