A Simulation-based Fault Injection Mechanism of Digital Circuit

Quan Zhou[1,a,*], Xin Yan[1,a] and Liang Yang[1,a]

[1]Xi'an Microelectronics Technology Institute, Taibai South Road, Xi'an, China
[a]zhouquan1234.ok@163.com
*corresponding author

Keywords: VHDL, test bench, fault-tolerance, reliability design, reliability validation, fault injection.

Abstract. Fault injection is a key step in the validation of fault-tolerant design. This paper introduces a novel simulation-based fault injection method. The proposed method is implemented directly in Test Bench (TB) by modifying the signal values in VHDL model, and the fault type and ratio, transient fault duration and fault injection rate etc. can be easily adjusted according to requirements. It also supports the random fault injection by giving a random distribution. Compared with existing approaches, this simple method that can be designed and used immediately has a better extendibility and a better tailing capability due to the fact that it is field programmable in TB. Fault injection experiment shows that this approach is flexible and easy to use.

1. Introduction

The performance of digital processor has been quickly improved with the increasing integrated level and decreasing feature size in recent years. However, because of its high integrated level, low threshold voltage and high clock frequency, the processor is more sensitive to crosstalk, electromagnetic interference and heavy-ion radiation etc. It is easy to arouse failures inside the chip [1], [2]. So, some reliability enhancement measures, generally including three levels: system level, circuit level and device level [3], are adopted to improve chip dependability and ensure each function carried out correctly as far as possible.

In system level strengthening, front-end designers take some architecture reinforcement measures to implement fault-tolerance [4], [5], [6]. Such design methods are mainly to resist permanent or transient faults caused by bad external environment. For the verification of fault-tolerance, it is necessary to inject failures and then check the operations to determine whether the processor still run correctly. The goal of this paper is to present a simulation-based fault injection method. This method on the principle of simplicity and high efficiency is easy to achieve and use.

The rest of the paper is organized as follows. Section 2 introduces the related works. In Section 3, the proposed fault injection mechanism is described. The fault injection experiments are depicted in Section 4, and the proposed method is evaluated in Section 5. Finally, we present conclusions in Section 6.

2. Related Works

In reliability demonstration, the dummy faults used to simulate the soft-errors caused by real environment are injected into a running system, after which an analysis of correctness is made to determine whether the adopted reliability mechanisms have the intending fault-tolerant capability. Fault injection techniques can be classified in three main categories: hardware fault injection, software fault injection and simulation-based fault injection [7], [8].

The simulation-based fault injection technique uses the VHDL model as the target system, and it usually includes two implementation methods: modifying the VHDL model and altering the value of signals and variables defined in the prototype [9].

The modification of VHDL model mainly uses the saboteur or mutation method. It is necessary to add two kinds of dedicated components called “saboteur” and “mutant” which are usually inactive during normal system operation and activated only to inject faults [7], [8], [9], [10]. A saboteur is a special VHDL component added to the original model. The mission of this component is to alter the value and/or timing characteristics of one or more signals when a fault is injected, and it remains inactive during the normal operation of the system [8]. A mutant is a component which replaces another component. While inactive, it works like the original component, but when activated, it behaves like the component in presence of faults [9].

The saboteur and mutant can be designed using the full strength of VHDL language. Therefore, this approach can support a wide range of fault model that can be expressed within the VHDL semantics. By analyzing the drawbacks of some models of existing saboteurs and mutants, J. C. Baraza et al. give out an enhancement of fault injection techniques based on the modification of VHDL code [11], and propose some new models which can be automatically inserted into a model in order to perform a fault injection campaign. A typical application paradigm of modification of VHDL-model method, S. Nimara et al. come up with four types of saboteurs and analyze the impact of probabilistic faults in interconnects by means of HDL saboteur-based simulated fault injection using Wishbone bus as the target system [12]. However, the saboteur or mutation introduces redundant modules and a lot of control signals, and is of high spending and complex in realization.

Compared with saboteur or mutant, the approach of altering the values of signals and variables has the advantages that there is no redundant components are introduced [9], [13]. Because its implementation is relatively simple, it is more suitable for using at the earlier stage of the dependability design, achieving earlier validation and evaluation of reliability mechanisms [9], [14]. The technique of altering the signal values is essentially to set a fault value for signals defined in VHDL-model, usually including stuck-at-0/1, open lines, indetermination, bit-flip, and pulse. D. Gil et al. think that the simulations of the four fault models, including stuck-at, bit-flip, indetermination, and delay, can represent transient physical faults of different types: transient in power supply, crosstalk, electromagnetic interferences, temperature variation, αradiation and cosmic radiation. The physical faults above can directly vary the values of voltage and current of the logical levels of circuit nodes. Therefore, they modify the values of the signals and variables using a simulation-based technique to simulate the above four fault models, developing a fault injection tool. Some of their study and related experimental results are shown in [15], [16]. In [17], a simulation-based fault injection technique is adopted to inject transient and permanent single and multi-bit faults by forcing the signal to a selected fault value using the Force and Release statements(the features in the VHDL 2008 standard). A fault injection unit is developed to inject faults into a VHDL target system using force and release assignments. However, unlike the proposed method, this unit is still needed to be added to the VHDL source code in the set up phase to create a mutated VHDL source code. In our work, an altering signal method by Force is also introduced. The detailed comparisons of the proposed method and [17] are described in Section 5.

So far, the fault injection techniques aiming at VHDL model have been widely studied and many systematic tools also have been proposed, such as MEFISTO [9]，VFIT [13]，MEFISTO-L [18]，VERIFY [19]. MEFISTO utilizes the simulator built-in commands to alter the values of signals and variables, thus the realization of fault injection. The subsequent version MEFISTO-L realizes the injection campaign by inserting fault injection components. VFIT is also a fault injection tool aimed at VHDL model and the three techniques of simulator built-in command, saboteur and mutant are adopted in its implementation. VERIFY achieves the fault injection by adding the fault behavior description to the elementary components (such as NOT-gate) that can be executed directly by simulator during the fault injection campaign. All of these tools attain fault injection aiming at VHDL models from different perspectives, and from a certain extent, can automatically trace and further analyze the operation conditions of system with the present of injected failures. However, it is the systematicness and comprehensiveness which make the fact that a lot of manpower, more material resources, and a longer development cycle are needed to realize such tools.

Actually, during the prophase of design, it is unable or unnecessary to use such a tool to validate the availability of fault-tolerant strategies. A key process in the early verification is the realization of fault injection. That is to give out a method for fault injection in a simple and practicable, economical and efficient way, and ensure that the results caused by injected errors can be analyzed. Given all this, this paper presents a fault injection technique that is implemented directly in Test Bench by modifying the values of signals and variables in VHDL model. The notable advantages of this method over existing approaches are simple and efficiency. With the capability of field realization and be used immediately, it improves the speed of depending design and verification, and has a better extendibility and a better tailing capability comparing with the existing approaches.

3. Simulation-based Fault Injection in Test Bench

The proposed method performs fault injection with a short description in TB file, and its block diagram is shown in Fig.1. This mechanism includes four parts: Target Signal Set (S_n), Fault Generator, Fault Injector, and VHDL Model. The S_n is created based on VHDL Model and there is no information interaction between S_n and VHDL Model during the simulation time, hence the dot line between them in Fig.1.

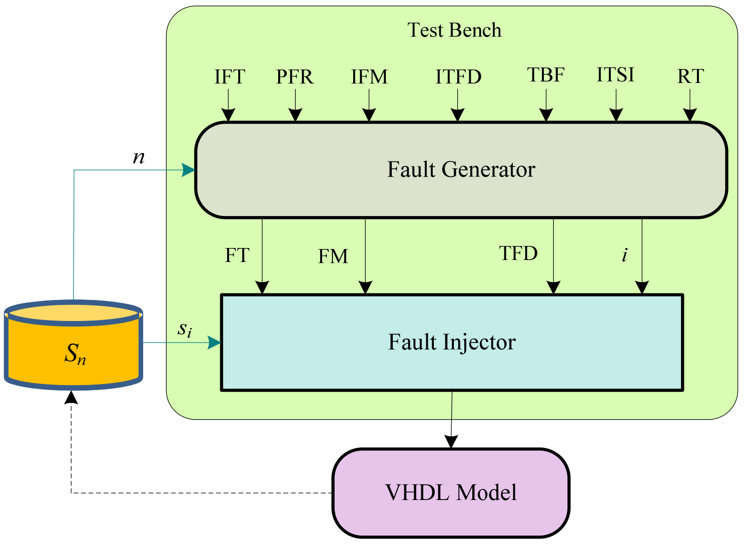


Figure 1 Simulation-based fault injection mechanism in TB.

3.1. Target Signal Set

The fault injection mechanism based on TB simulation needs firstly to extract the failure points on which the signals can make up the S_n. Any signals can be added to S_n as long as a failure needs to be activated on it. S_n can be expressed as follows:

$$S_n = \{s_1, s_2, s_3 \cdots s_n\}$$

where n indicates the total number of elements of set S_n, and the elements s_i ($1 \leq i \leq n$) actually represent the modularization index path of a target signal in the hiberarchy through which a specific target signal can be referenced. S_n can be created as a file in which each element s_i occupies a line for the convenience of element indexing.

3.2. Fault Generator

The proposed method can simulate two fault types: permanent fault and transient fault. The permanent fault includes four models: stuck-at-0, stuck-at-1, open line and indetermination. The transient fault includes three models: bit flip, pulse and indetermination.

As shown in Fig.1, the inputs of Fault Generator involve the value of n obtained from S_n and the initial informations specified before the starting of simulation. The initial informations are presented as follows.

Initial Fault Type (IFT): can be specified as Permanent or Transient for nonrandom fault injection.

Permanent Fault Rate (PFR): can be specified as a real number in the range of [0, 100] that denotes the proportion of permanent fault in the random fault injection campaign. For example, if PFR is specified as a, the proportion of permanent fault is a%, and that of transient fault is (100-a)%.

Initial Fault Model (IFM): can be specified as one of the fault models (stuck-at-0, stuck-at-1, open line, bit flip, pulse and indetermination) according to the specified IFT for nonrandom fault injection.

Initial Transient Fault Duration (ITFD): can be specified as a range from which the number of clock cycle that the transient fault last for is achieved. For example, if ITFD is specified as [0.01, 10], a real number is achieved uniformly and randomly from the range of [0.01, 10], then multiplied by clock period time, and the product time is taken as the duration of transient fault. If IFT has been specified as Permanent, ITFD is unavailable and the permanent fault is active until the end of simulation.

Time Between Failures (TBF): can be specified as a range from which the number of clock cycle between two fault injections is achieved. For example, if TBF is specified as [100, 150], a real number is obtained uniformly and randomly from the range of [100, 150], then multiplied by clock period time, and the production time is taken as the time between two failures. Changing the lower and upper limits of this range can achieve the goal of adjusting the mean time between two failures. The smaller the upper limit is, the shorter the mean time between two failures, and the higher the density with which faults are injected.

Initial Target Signal Index (ITSI): can be specified as an integer in the range of [1, n] that denotes the element index in S_n for nonrandom fault injection.

Random Type (RT): can be specified as a random distribution that determines the fault model and target signal index in random fault injection. In our application, RT can be specified as Nonrandom or Uniform_Distribution (the more random distributions can be further added). Nonrandom actually indexes none of the random type is used in fault injection campaign, and a fault is injected into the system according precisely to the initial information after each execution of this injection mechanism. Uniform_Distribution means that the fault model and target signal index are generated based on uniform distribution, thus the random fault injection achieving. At this time, the input informations IFM and ITSI are neglected.

The function of the Fault Generator is according to the RT to generate the control signals of Fault Injector. Working in real environment, the processor failure usually has the characteristics of some kind of randomness. Using the random functionality of the proposed Fault Generator, a better simulation of the random characteristics of soft errors is obtained in a simple way. Simulating the random faults by designing corresponding random distribution procedure is a notable advantage of the proposed method.

3.3. Fault Injector

The fault injector receives the information coming from Fault Generator, including Fault Type (FT), Fault Model (FM), Transient Fault Duration (TFD) and target signal index (i), and does the fault injection operations.

The Fault Injector firstly takes the target signal out from S_n according to index i and sets it to a specified failure value by force, thus the implementing of fault injection. It is necessary to perform different operations by different fault model during the fault injection campaign. The pseudo-code description of Fault Injector in QuestaSim simulator is shown in Fig. 2. The calls “Init_Signal_Spy( )” and “Siganl_Force( )” that implement the signal mirroring and force a signal to a specified value respectively are the library functions provided by QuestaSim simulator [20]. In Fig. 2, the value of the target signal (TgtSignal) is mirrored onto the signal “Sig” defined in TB, and then the target signal value in normal operation of the processor can be obtained by checking the Sig. By inverting the normal value of Sig and assigning it to the target signal, the simulations of pulse and bit flip can be realized.

```
IF FaultType=Permanent Then
 Case FaultModel Is
    When Stuck_at_0 =>
      Signal_Force (TgtSignal, “0” , Now, Freeze, Open, 1);
    When Stuck_at_1 =>
      Signal_Force (TgtSignal, “1” , Now, Freeze, Open, 1);
    When OpenLine =>
      Signal_Force (TgtSignal, “Z” , Now, Freeze, Open, 1);
    When Indetermination =>
      Signal_Force (TgtSignal, “X” , Now, Freeze, Open, 1);
 End Case;
End IF;

IF FaultType=Transient Then
 Init_Signal_Spy(TgtSignal, “/TB/Sig” , 1);
   Case Sig Is
     When ‘0’ => Nsig := “1” ;
     When ‘1’ => Nsig := “0” ;
   End Case;
   Case FaultModel Is
     When Pulse =>
       Signal_Force (TgtSignal, Nsig, Now, Freeze, Now + FaultDuration, 1);
     When BitFlip =>
       Signal_Force (TgtSignal, Nsig, Now, Deposit, Open, 1);
     When Indetermination =>
       Signal_Force (TgtSignal, “X” , Now, Freeze, Now +FaultDuration, 1);
   End Case;
End IF;
```

Figure 2 Pseudo-code description of Fault Injector in QuestaSim simulator.

It is worth noting that the way of signal mirroring and force assignment may be different according to different simulator. The “Init_Signal_Spy( )” and “Signal_Force( )” are provided by QuestaSim simulator, and however “Signal_Agent( )” and “Force( )” are used in the simulator

Active-HDL. In VHDL 2008 standard, new features were introduced to VHDL language. One of these features is signal assignment with force and release commands used in a sequential statement in the VHDL description [21], thus greatly reducing the dependence on simulation tools of the proposed method.

4. Fault Injection Experiments

Fig. 1 shows a complete set of fault injection mechanism that can be field designed in TB and put into use immediately to validate and evaluate the reliability design. By far, in our research team, the proposed method has been widely used in the design of Revealer1601prh processor, the National Science and Technology Major Project. For the validation and evaluation of fault-tolerant strategies, the designer of each module only needs to create the S_n and set the initial parameters of Fault Generator before the starting of simulation.

The description of the applicability of the proposed method is presented below by fault injection experiments in which a timer/performance-counter of coprocessor of Revealer1601prh is taken as a target module.

4.1. Configurable Timer/Performance-counter

The Revealer1601prh processor is a multi-core chip based on network-on-chip of which the PowerPC470 as the master core, sixteen homogeneous high performance DSPs as slaver cores and a variety of peripheral controllers are inside. Each slaver core totally includes 9 ways configurable timer/performance-counter, as shown in Fig.3. The Timer0 to Timer7 can be used as a timer or performance-counter respectively by configuring the control register file. If configured as a timer, the value of internal counter will add 1 at each clock cycle, thus the function of timing. If configured as a performance-counter, the counter counts the core's inner events that inputted from the interface of Pf_Source (256 event sources are supported). In addition, all of the Timer0 to Timer7 can generate interrupt signals outputted from InterruptOut.

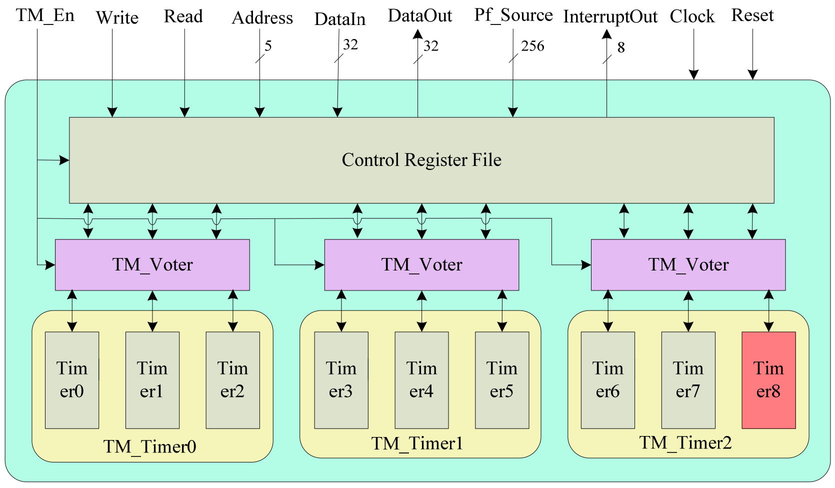


Figure 3 Configurable timer/performance-counter architecture.

Another notable advantage of this configurable timer/performance-counter is that it can work in triplication fault-tolerant mode according to the index of inputted signal TM_En. An additional Timer8 invisible to users is therefore introduced for the purpose of triple modular fault-tolerant. When TM_En is enabled, the timer/performance-counter is reinforced by binding each three timers of the nine ways, thus the total three triple-modular timers (TM_Timer0 to TM_Timer2) are provided. The input and output of each timer are voted by triple modular voter (TM_Voter). If TM_En is invalid, all the Timer0 to Timer7 work in single mode independently without the capability of fault-tolerance.

4.2. Fault Injection Experiments Based on Timer/Performance-counter

In this section, the fault injection campaigns based on the triplication working mode (TM_En is enabled) of the configurable timer/performance-counter are presented. In the experiments, TM_Timer1 is configured as a performance-counter. Because TM_Timer0 and TM_Timer2 are not started during all the injection campaign, only TM_Timer1 is taken as the target module. That is to say the S_n only includes all the signals defined in TM_Timer1 and the practice shows that n=1213. In our experiments, the performance event counted by TM_Timer1 is "the number of instructions the core executed". The simulation platform is QuestaSim simulator. The experiments are divided into two categories, one of which evaluates the reliability of timer/performance-counter without the permanent fault present and the other has the same simulation conditions excepts that a random permanent fault is introduced.

4.2.1. First Category Experiment

The first category experiment only sets one injection mechanism. The experiment conditions are described as follows:

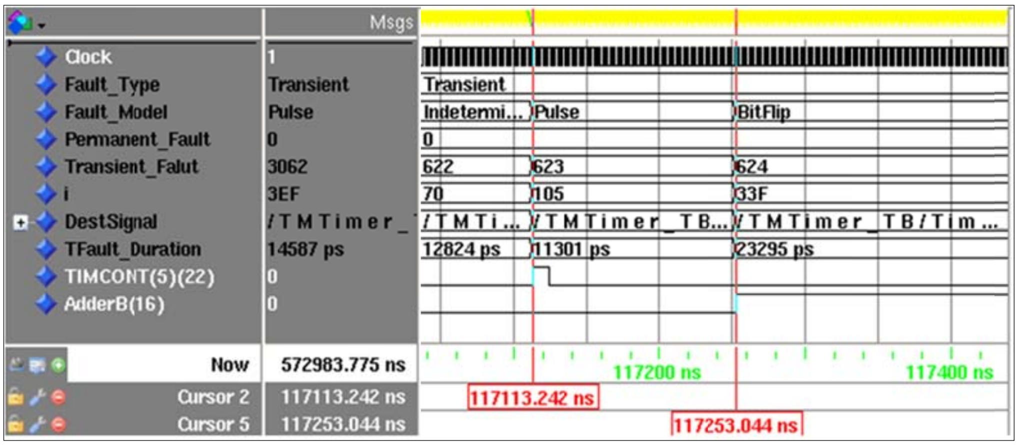
1. The RT is specified as Uniform_Distribution.
2. Only the transient faults are injected into the target system, no permanent fault exists (the PFR is specified as 0).
3. "The number of instructions the core executed" is taken as the event source. One thousand program segments are executed by core, and that the number of instructions in each program segment is unfixed in the range of 100 to 300.
4. Two sets of ITFD are used. Namely, transient fault duration is set to [0.01, 10]T and [10, 80]T respectively where T is the system clock cycle and T=2.5 ns. (The ITFD is specified as [0.01, 10] and [10, 80] respectively.)
5. Seven sets of TBF are used corresponding to each set of ITFD respectively. (The TBF is specified as [0, 50], [50, 100], [100, 150], [150, 200], [200, 250], [250, 300], and [300, 350] respectively.)

The injection procedure starts to run when the TM_Timer1 begin to work. One thousand performance counts, each of which counts the number of instructions in a program segment, are performed by TM_Timer1. Comparing each counter result achieved from TM_Timer1 with the actual number of instructions of each program segment, we can get the malfunction times of TM_Timer1 with the transient fault present, and then the failure rate is computed. The running details can be checked by simulation waves and printed trace files. Fig. 4 shows a wave segment in a certain period of time, in which a transient fault with the fault model of pulse lasting for 1301ps is activated on the bit signal TIMCONT(5)(22) at the simulation time 117113.242ns and a bit flip is activated on the bit signal AdderB(16) at 117253.044ns.

Fig. 5 shows the variation of failure rate with the TBF. By Fig.5, for a fixed TBF, the longer the transient fault duration is, the higher the failure rate, and with the increasing of TBF, a decreasing failure rate results.

We extend the performance-count task by increasing the number of instructions the core executed in each program segment to 300 to 600 and the other conditions remains, and the test results are shown in Fig. 6.

A comparison of Fig. 5 with Fig. 6 reveals that, with the same transient fault duration and the same TBF, the longer it takes a component of the processor to complete a continuous task, the higher the failure rate.



	Msgs
Clock	1
Fault_Type	Transient
Fault_Model	Pulse
Permanent_Fault	0
Transient_Falut	3062
i	3EF
DestSignal	/TMTimer_
TFault_Duration	14587 ps
TIMCONT(5)(22)	0
AdderB(16)	0
Now	572983.775 ns
Cursor 2	117113.242 ns
Cursor 5	117253.044 ns

Transient
Indetermi... Pulse BitFlip
0
622 623 624
70 105 33F
/TMTi... /TMTimer_TB... /TMTimer_TB/Tim...
12824 ps 11301 ps 23295 ps
117200 ns
117400 ns
117113.242 ns
117253.044 ns

Figure 4 Simulation wave segment in the fault injection campaign.

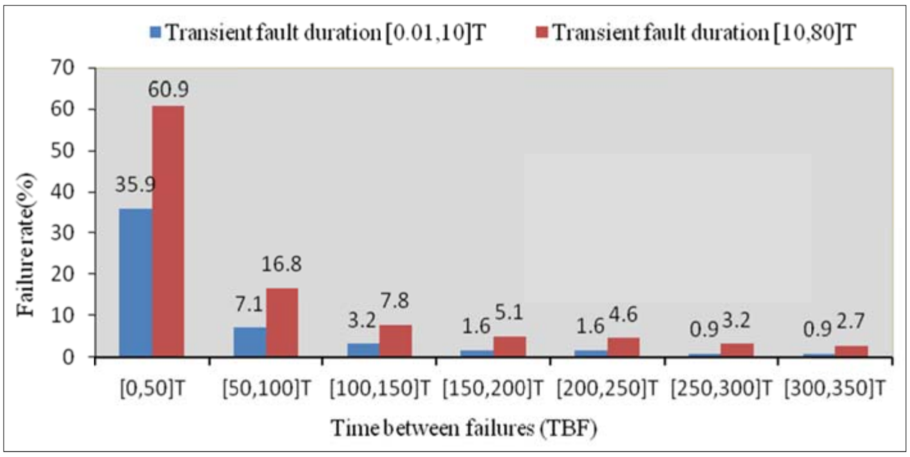


Figure 5 Failure rate with the transient fault duration [0.01, 10]T and [10, 80]T. The instruction number of each program segment is in the range of [100, 300] and no permanent fault exists.

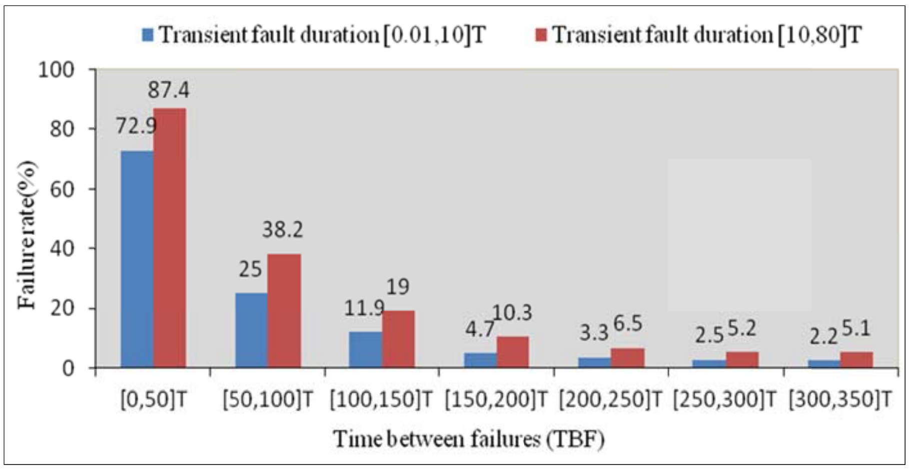


Figure 6 Failure rate with the transient fault duration [0.01, 10]T and [10, 80]T. The instruction number in each program segment is in the range of [300, 600] and no permanent fault exists.

4.2.2. Second Category Experiment

In real applications, with the time going, permanent fault(s) may occur inside the processor. So, the second category experiment is conduced to evaluate the effect of permanent fault on the configurable timer/performance-counter. The second category experiment is the same as the first one excepts that an additional injection mechanism is introduced to inject a random permanent fault and then is suspended due to the appendent "wait" statement at the end of the process.

Given the two different lengths of the performance-count tasks, the second category experiment results are shown in Fig.7 and Fig. 8, respectively.

Comparing Fig. 5 and Fig. 7, or Fig.6 and Fig. 8, we can find the fact that a component with permanent fault present has evidently higher failure rate than an intact one, thus a lower reliability. Besides, we can still find that the failure rate decreases with the increase of TBF in both the comparisons.

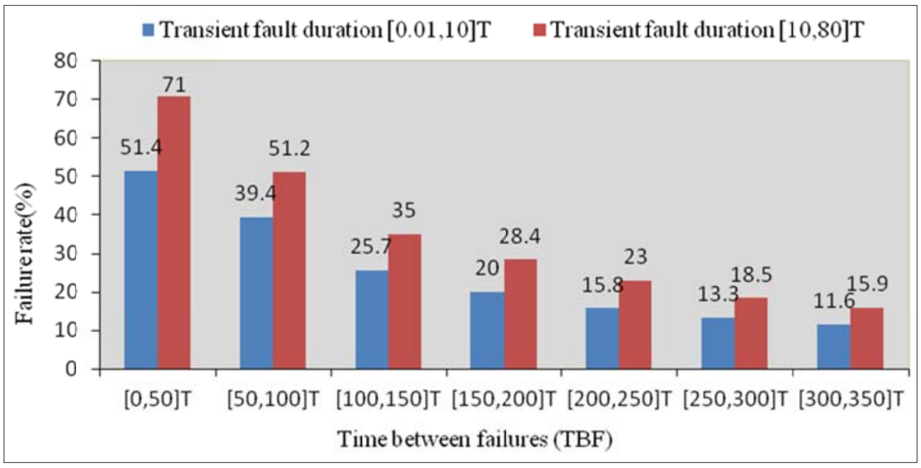


Figure 7 Failure rate with the transient fault duration [0.01, 10]T and [10, 80]T. The instruction number in each program segment is in the range of [100, 300] and a random permanent fault exists.

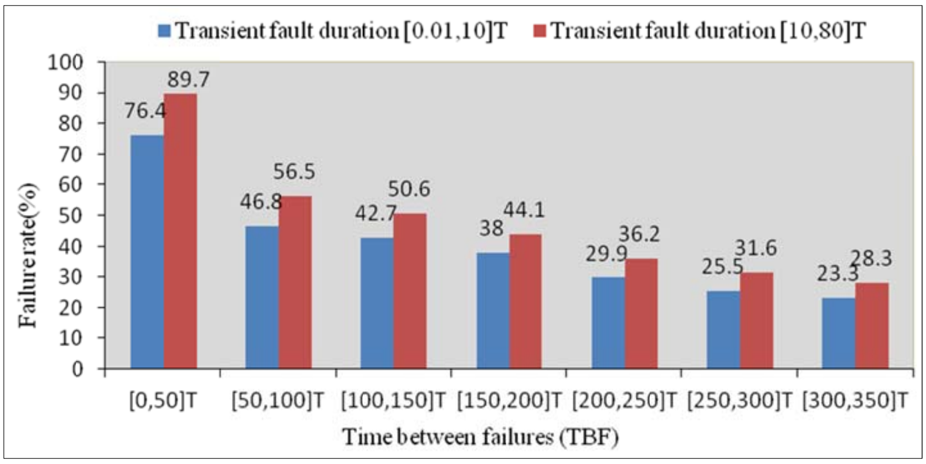


Figure 8 Failure rate with the transient fault duration [0.01, 10]T and [10, 80]T. The instruction number in each program segment is in the range of [300, 600] and a random permanent fault exists.

5. Evaluation of The Proposed Method

The main contribution of this work is the simple fault injection technique which is realized directly in TB. The fault injection campaign shows the proposed method is easy to use and has a good applicability. In our work, the force assignment is used to inject faults into a VHDL system and a fault injection procedure is realized with the QuestaSim simulator.

There are two kinds of ways to realize force assignment. The first one is calling the function that is provided by simulator, such as the one used in this paper. However, this method is simulator-dependent. The other way without dependent on simulator is to use the Force and Release statements of the VHDL 2008 standard, as it does in [17]. In order to further explain the differences between [17] and the proposed method, a detailed comparison of [17] and the proposed method is shown in Table Ⅰ.

By Table Ⅰ, although the force assignment fault injection method is adopted in both the mechanisms, a clear distinction between these two approaches exists. In [17], the fault injection unit (FIU) is developed to implement fault injection using force and release assignments. The user specifies the FIUs and all the FIUs are inserted into the source code before the compiling phase. The distributed force assignment method is adopted and the FIU is inserted at each injection point in [17]. However, a centralized force assignment in TB is realized in the proposed method, so it does not need to preprocess for VHDL model. This improvement is conducive to further implement rapid fault injection.

Besides, because the proposed method is directly programmed in TB, it is very easy to realize the random injection that is not supported in [17]. Generally speaking, the proposed fault injection mechanism has a lower implementation cost than [17].

Table 1 Comparison of [17] and the proposed method

	VHDLSFI in [17]	**The proposed method**
Fault injection	Distributed force assignment	Centralized force assignment in TB
Initial configuration	Parameters initialization is needed	Parameters initialization is needed
Fault model	Single and multiple. Permanent: tuck-at-0/1, bridging, open lines, indetermination. Transient: bit-flip, pulse, indetermination.	Single and multiple. Permanent: tuck-at-0/1, open lines, indetermination. Transient: bit-flip, pulse, indetermination.
System input	VHDL model and object signals	VHDL model and object signals
Random injection	Nonsupport	Support
Preprocess of VHDL model	Required (to insert FIU to create a mutated VHDL source code, generate a simulator macro)	Not Required (directly run the simulation)
Analysis of results	Analysis of results based on trace files.	Analysis of results based on trace files and wave froms.
Implement cost	Higher	Lower

6. Conclusions

Fault injection is a critical step in dependability verification. This paper presents a simulation-based fault injection mechanism that is directly realized in TB. This method simple in achievement only needs a short description in TB for a random or nonrandom injection campaign. The injection experiments aiming at a configurable timer/performance-counter show that the presented method is very flexible and easy to use due to the assignable initial parameters by which specify the fault type and rate, adjust the transient fault duration, and set the time between failures to control injection rate. Besides, it has a better extendibility and a better tailing capability compared with the common approaches because it is field programmable in TB.

References

[1] M. C. Casey, B. L. Bhuva, S. A. Nation, et al. “Single-event effects on ultra-low power CMOS circuits,” IEEE Int. Reliability Physics Symp., pp. 194-198, Apr. 2009.
[2] R. Garg, S. P. Khatri, “3D simulation and analysis of the radiation tolerance of voltage scaled digital circuit,” IEEE Int. Conf. Computer Design, pp. 498-504, Oct. 2009.
[3] L. Wang, G. H. Zhang, Y. L. Zeng, et al, “Low power and high write speed SEU tolerant SRAM data cell design,” Science China Technological Sciences, vol. 58, no. 11, pp: 1983–1988, Nov. 2015.
[4] M. Nicolaidis, “Time redundancy based soft-error tolerance to rescue nanometer technologies,” Proc of 17th IEEE VLSI Test Symp., pp: 86-94, Apr. 1999.
[5] R. V. Kshirsagar, R, M, Patrikar, “A novel fault tolerant design and an algorithm for tolerating faults in digital circuits,” IEEE 3rd Int. Design and Test Workshop, pp: 148-153, Dec. 2008.
[6] H. Farbeh, S. G. Miremadi, “PSP-cache: a low-cost fault-tolerant cache memory architecture,” Proc of the Conf. on Design, Automation & Test in Europe, no. 164, Mar. 2014.
[7] N. Song, J. Qin, X. Pan, et al, “Fault injection methodology and tools,” IEEE Int. Conf. on Electronics and Optoelectronics (ICEOE), vol. 1, pp. 47-50, Jul. 2001.
[8] J. Gracia, J. C. Baraza, D. Gil, et al, “Comparison and application of different VHDL-based fault injection techniques,” Proc. of IEEE Int. Symp. on Defect and Fault Tolerance in VLSI Systems, pp. 233-241, Oct. 2001.
[9] E. Jenn, J. Arlat, M. Rimen, et al, “Fault injection into VHDL models: the MEFISTO tool,” Digest of Papers, IEEE Twenty-Fourth Int. Symp. on Fault-Tolerant Computing, pp. 66-75, Jun. 1994.
[10] J. C. Baraza, J. Gracia, D. Gil, et al, “Improvement of fault injection techniques based on VHDL code modification,” Tenth IEEE Int. High-Level Design Validation and Test Workshop, pp. 19-26, Dec. 2005.
[11] J. C. Baraza, J. Gracia, S. Blanc, et al, “Enhancement of fault injection techniques based on the modification of VHDL code,” IEEE Trans on Very Large Scale Integration (VLSI) Systems, vol. 16, no. 6, pp. 693-706, Jun. 2008.
[12] S. Nimara, A. Amaricai, O. Boncalo, et al, “Probabilistic saboteur-based simulated fault injection techniques for low supply voltage interconnects,” IEEE 10th Conf. on Ph. D. Research in Microelectronics and Electronics (PRIME), pp. 1-4, Jun. 2014.
[13] J. C. Baraza, J. Gracia, D. Gil, et al, “A prototype of a VHDL-based fault injection tool: description and application,” Journal of Systems Architecture, vol. 47, no. 10, pp. 847-867, Apr. 2002.
[14] D. Gil, J. Gracia, J. C. Baraza, et al, “Study, comparison and application of different VHDL-based fault injection techniques for the experimental validation of a fault-tolerant system,” Microelectronics Journal, vol. 34, no. 1, pp. 41-51, Jan. 2003.
[15] D, Gil, J. V. Busquets, J. C. Baraza, et al, “A fault injection tool for VHDL models,” Fastabs of the 28th Fault Tolerant Computing Symp.(FTCS-28), Jun.1998.
[16] D. Gil, J. Gracia, J. C. Baraza, et al, “A study of the effects of transient fault injection into the VHDL model of a fault-tolerant microcomputer system,” Proc. of 6th IEEE Int. On-Line Testing Workshop, pp.73-79, Jul. 2000.
[17] F. Pournaghdali, A. Rajabzadeh, M. Ahmadi, “VHDLSFI: A simulation-based multi-bit fault injection for dependability analysis,” IEEE 3th Int. Conf. on Computer and Knowledge Engineering (ICCKE), pp. 354-360, Oct. 2013.
[18] J. Boué, P. Pétillon, Y. Crouzet, “MEFISTO-L: A VHDL-based fault injection tool for the experimental assessment of fault tolerance,” Digest of Papers, 28th Annual Int. Symp. on IEEE Fault-Tolerant Computting, pp. 168-173, Jun. 1998.
[19] V. Sieh, O. Tschache, F. Balbach, “VERIFY: Evaluation of reliability using VHDL-models with embedded fault descriptions,” Digest of Papers, 27th Annual Int. Symp. on IEEE Fault-Tolerant Computing, pp. 32-36, Jun. 1997.
[20] Mentor Graphics Corporation, “Questa SIM User’s Manual,” 1991-2011.
[21] IEEE Standard VHDL Language Reference Manual, 2008.