Fault Simulation and Formal Analysis in Functional Safety CPU FMEDA Campaign

Xueying Yang*, Dongyan Zhao, Yichu Jiang, Xige Zhang, Yidong Yuan
Beijing Smart-Chip Microelectronics Technology Co., Ltd. Beijing, China

*Corresponding author’s email: yangxueying@sgitg.sgcc.com.cn

Abstract. In accordance with safety requirements of industrial control, a functional safety CPU is designed targeting controller IC used in State Grid. Functional safety verification flow based on FMEDA is setup for the project, which totally comply with IEC61508. In this paper, fault injection with fault simulation and formal analysis flow of functional safety verification is introduced in detail, which is set up for calculation of diagnose coverage on random hardware failure. Employment of formal method completes 2-3 weeks fault analysis in 52 hours, which improved diagnose coverage convergence.

Keywords. Functional safety, FMEDA, fault simulation, formal analysis, diagnostic coverage

1. Introduction
In recent years, with the development of the automotive industry and the maturity of autonomous driving technology, the concept of functional safety has gradually become familiar in the field of integrated circuit design. In fact, functional safety has always been essential in industrial and automotive electronics as a design requirement on system level. But nowadays as system scale is getting bigger and bigger, the response speed, cost, system complexity, and versatility are all facing great difficulties in the design of functional safety on system. At the chip level, especially at the core of the control chip—the addition of functional safety design at the CPU level has gradually become a trend [1]. This project is a functional safety CPU for power grid control. The Safety Integrity Level (SIL) target is SIL3, which is the highest level that a single chip can achieve. According to IEC61508-2 [2], Safe Failure Fraction (SFF) needs to be greater than 99% for Type B devices reach SIL3. With reference to Appendix C of the standard, the diagnostic coverage (DC) also needs to be higher than 99%. Bellotti et al. compared several common CPU designs for functional safety in [1]. We chose the dual-core lockstep structure, which has the highest theoretical diagnosis coverage, combined with tightly coupled data memory (CCM) data error correction and bus integrity protection (including master and slave interfaces). The structure is shown in Figure 1.
2. Fmeda based functional safety verification

The diagnostic coverage in IEC 61508-4 is defined as "fraction of dangerous failures detected by automatic on-line diagnostic tests". In formula (1), DC is the diagnostic coverage, \( \sum \lambda_{DD} \) is the number of detectable faults, and \( \sum \lambda_{D\text{total}} \) is the total number of faults.

\[
DC = \frac{\sum \lambda_{DD}}{\sum \lambda_{D\text{total}}}
\]  

Based on FMEDA, the diagnosis coverage (DC) calculation for random hardware failures (Random Hardware Failure) is a necessary step for functional safety projects. For a complex design such as a CPU, traditional manual analysis and calculation of fault coverage are obviously not feasible, and functional safety verification is required to collect and calculate coverage.

IEC 61508-2 Table A.1 shows the failures that need to be considered when calculating the diagnostic coverage of common IC types. For the CPU, 1) registers and embedded RAM, 2) encoding, decoding and operation, including flag register, 3) Address calculation, 4) PC and stack pointer.

The concept of coverage is the same as DFT. In the verification, fault injection and fault simulation or formal verification are performed on the safety-related parts of the functional safety chip according to different fault types. It can be determined whether each fault point can be effectively detected by the safety mechanism. Therefore, a reliable diagnostic coverage value can be obtained.

2.1 Fault model based fault simulation

A fault simulation flow setup by ARM for functional safety shown in [3]. Tools supporting fault simulation include Synopsys’ Z01X [4], Cadence’s Incisive, etc. There is little difference in efficiency between the tools, and they all support front and back simulations. The fault types support fixed faults and transient faults. In the functional safety verification of this article, we chose Synopsys' Z01X.

1). Fault model

Environmental impacts, circuit aging, etc., caused by short-circuits and open circuits to power or ground, transistor breakdown, interconnection or disconnection between nodes, etc., physical failures that occur in actual chips can be simulated by abstract models. Similar to DFT, the fault models that need to be considered in functional safety verification are mainly divided into stuck-at faults and transient faults.

a). Stuck-at fault

Stuck-at fault means that the fault point is fixed to 1 (SA1) or 0 (SA0) and cannot be reversed. According to the location, it can be divided into 1) output SA0/SA1, 2) input SA0/SA1, 3) node SA0/SA1, as shown in Figure 2.
b). Transient fault
In a transient fault, the fault state at the fault point occurs only once and disappears after a certain period of time. In reality, environmental influences, ionizing radiation, or malicious attacks can cause transient failures. Transient faults occur at random times, and whether they cause system failure depends on whether the faults propagate downwards within the effective time. Transient fault injection only considers registers. Different from stuck-at faults, transient fault injection requires not only its location and status, but also the time of fault injection.

2). Fault simulation
In the fault simulation, the simulation tool establishes two models, the original design is the GM (Good Machine) model, and the fault injection is the FM (Fault Machine) model. For each fault, the tool compares the observation points and detection points of the two models. As shown in Figure 3, the difference between GM and FM values is observed (observed)/detected (detected). A fault that cannot be propagated to the observation point under any conditions is regarded as a safety fault. A fault that can propagate to the observation point but cannot be effectively detected by the safety mechanism is a dangerous fault, otherwise it is a safe fault.

3). Results
Through fault simulation, the faults in the design are classified according to their states. Except for the detected safety faults and the dangerous faults that cannot be detected, some fault states are marked as NS (Not Strobed), NO (Not Observed), NC (Not Controlled), respectively indicate that under all incentives provided
a) The fault state can be transferred to the observation point but not sampled (NS)
b) The fault is not observed at the observation point (NO)
c) The fault point cannot be reversed (NC)
These fault status are due to insufficient stimulus. Traditionally, manual analysis on design source
code are required. Direct test cases are created to meet coverage requirements.

2.2 Formal method

Regarding formal verification (FV) [5], there is the following definition: FV is the use of tools to mathematically analyze all possible states in a design, rather than the result of calculating a specific state value. Formal verification is aimed at "correctness", "completeness", and "corner cases", which coincides with the purpose of functional safety [6]. In this paper, with the help of formal verification methods, the calculation efficiency of fault diagnosis coverage is greatly improved.

1). Formal verification and functional safety

Formal verification is generally divided into model checking, equivalence checking and theorem proof. Functional safety verification, as an application of model checking, uses an exhaustive method of all state spaces to conduct assertion checks at observation points and detection points for each fault.

2). Types of formal analysis method

Formal verification tools analyze the fault tree to know whether a random hardware fault is observed and detected in all possible state spaces. Specific analysis methods include structured analysis, controllability analysis, observability and detectability analysis. Combine and apply various analysis methods to achieve fault coverage calculation.

a). Structural analysis

Structured analysis based on COI (Cone of Influence) is used to find faults that are not on the path of the observation point and mark them as safe faults, as shown in Figure 4. In the Figure, 5 faults are on the path of observation point 0, 6 faults are on the path of observation point 1, and the other 3 faults are not on the path of any observation point.

![Figure 4. Structural Analysis](image)

b). Controllability analysis

If the 0/1 state of the fault point cannot be reversed to another state, the fault at this point is uncontrollable and is a safe fault. As shown in Figure 5, because the input is fixed to 0, the output point is actually impossible to flip to 1, so its SA0 failure will not cause failure.

![Figure 5. Controllability Analysis](image)

c). Observability analysis and detection analysis
Comparison performed between GM and FM to see if faults can be propagated/observed at the observation/detection points. If the fault state cannot cause the observation point/detection point to be different on the GM and FM side, it is a safe fault. If it can propagate to both observation point and detection point, it is safe too. If the fault can be seen on observation point and cannot cause different status on detection points, it is a dangerous fault, as shown in Fig. 3.

Among the three types of analysis above, only observability analysis and detection analysis require formal calculations, which are time-consuming and computationally intensive.

3) FV tools
Cadence’s Jasper Gold, One Spin’s 360DV-Verify, Synopsis’s VC Formal, and Mentor’s Questa Formal all provide formal verification tools for functional safety. In addition, local EDA company Orcas Microelectronics Technology also released AveMC. In this paper, we chose Synopsis’s VC Formal [7].

3. Case study

3.1 Fault simulation with Z01X
In the failure modes of the functional safety CPU in this paper, fault numbers related to interface failure modes are relatively small, and can be implemented in the UVM functional verification environment. The scope of the core failure mode is the entire main core, including a large number of registers and nets. We chose to use the Z01X tool for fault simulation. The simulation environment is shown in Figure 6.

To use Z01X for fault simulation, we first make the following preparations:
1) The design to be tested, including Verilog tasks to distinguish GM and FM, function $fs\_compare in Z01X is used to compare the fault observation point and fault detection point of GM/FM;
2) Standard Fault Format (SFF) file, which defines the fault injection scope, fault types, and diagnostic coverage calculation formula corresponding to each failure mode;
3) Script file, used for Z01X simulation process control;
4) Stimulus. Although Z01X supports UVM TB, the interaction between DUT and TB will bring a greater loading. Relatively speaking, a vcd format file that only saves the waveform of in/out is a better choice.

In actual operation, the entire main core is used as the fault injection range, and the specific settings are as follows:
1) The fault type is Stuck-at faults (SA faults);
2) The fault injection points include the input of all sub-modules of the structured design;
3) Select PC value, stack pointer, operation and status flag, general register value, calculated address output as observation points;
4) The output signals Fault Indicator 1 and Fault Indicator 2 in Fig. 1 are fault detection points;
5) The test cases are the compliance testsuite and some unique function verification cases, totaling

![Figure 6. Z01X Flow](image-url)
2261, covering all function features.

Z01X compiles the design file and automatically generates a fault list. Within the scope we defined, the number of basic faults reached 21,359. After the first round of simulation, we get the results shown in Table 1.

Table 1. Diagnostic Coverage Result by Z01X

| Prime          | Total          |
|---------------|---------------|
| Total Faults  | 21359         | 30431         |
| Dangerous Diagnosed | 13280 | 18170 | 59.71% |
| Dangerous Not Diagnosed | 395   | 586  | 1.93%  |
| Safe          | 1936          | 2367          | 7.78%  |
| Dangerous Unobserved | 7684 | 11675 | 38.37% |
| Dangerous Assumed | 0   | 0   | 0%   |
| Diagnostic Coverage | 65.32% | 62.62% |

According to the detailed fault coverage information of each test case, it can be seen that out of 2261 test cases, only 145 vectors actually provide effective fault coverage, namely OD (observed detected)/ND (not observed diagnosed). Based on this, we can optimize the test cases and select only the valid parts for fault simulation.

On the other hand, we can see from the above report that there are still a large number of faults not covered, as shown in formula (2).

\[ \text{Remaining fault} = \text{DN} + \text{SU} + \text{DA} = 395 + 7684 + 0 = 8079 \]  

(2)

Traditionally, to increase fault coverage, it is necessary to conduct detailed analysis on RTL code and create direct cases manually. This requires a huge workload and is very inefficient.

3.2 Formal analysis with VCF

In order to solve the problem of analysis efficiency of uncovered failures above, we introduced a formal verification process.

The following definitions need to be made in formal verification:

1) Clocks, resets and reset process;
2) Observation point, detection point;
3) Fault list, we use the fault list output by Z01X as the input of VCF.

With the above definition, we get 8059 initial faults. Compared with the 8079 faults remained by Z01X, VCF ignores 20 clocks and reset signals as constraints. The tool will start to calculate from reset state.

Without other constraints, a structured analysis based on COI marked 2318 faults as safe faults. After controllability analysis, 19 faults were marked as safe. The two-step analysis does not require or only need very simple formal engine, and it is rarely time-consuming and very efficient.

Further analysis of the functional safety requirements shows that the debug module is only effective in debug mode. It should be in a debug mode forbidden during normal use, and we can constrain it. As debug mode constraint to be locked, structural analysis and controllability analysis are repeated, and the number of safety faults rises to 2,570. After 48 hours of observability and detectability analysis, 3258 faults were marked as safe, of which 688 were detectable faults.

In addition to the safety fault list provided by the tool’s automatic analysis, some faults require manual analysis. For example, the failure of the enable signal SA1 of the safety mechanism will not cause a dangerous failure and is regarded as a safety fault. There are some double-point faults, which are not within the scope of fault simulation. 472 such faults are manually removed. So far, as shown in Table 2, there are 4329 remaining 8079 faults after Z01X simulation. The uncovered faults have been reduced by 46%, and the effect is remarkable.
Table 2. Faults Covered by Z01X and VCF

|                   | Initial | Z01X | VCF |
|-------------------|---------|------|-----|
| Run Time (hour)   | 0       | 180  | 52  |
| Faults Covered    | 0       | 13280| 3750|
| Faults Remained   | 21359   | 8079 | 4329|

As can be seen from the above, with the help of formal verification tools, we can furtherly 1) increase analysis time, 2) add constraints, 3) add cutpoints to divide the design, 4) change the formal engine, etc., to achieve effective detection.

After effective fault filtering, continue to create direct cases, use Z01X simulation to cover, and pass the cycle of Z01X-->VCF-->Z01X to reach the final convergence of coverage.

4. Conclusion
In functional safety design, the diagnosis coverage of random hardware faults can be said to be the most important part of FMEDA. The functional safety verification based on fault injection provides us with reliable data and avoids excessive reliance on expert analysis. Fault simulation is intuitive and effective, but requires a lot of manpower and machine time, especially for the few complex faults left behind. It is a big challenge to create effective stimulus. With the improvement of algorithms and the improvement of calculation ability of computers, formal verification has become an emerging force in functional safety verification in recent years, which can analyze a large number of random faults with less labor cost.

For a complex design such as a CPU, due to its huge state space and a lot of sequential logic, it brings the problem of depth of analysis, which is an obstacle to formal verification. Reasonable segmentation of the design to localize fault detection as much as possible can not only increase coverage, but also speed up fault response time. Functional safety verification can provide a reliable guarantee for design improvement in this respect.

In this paper, on the basis of fault simulation, formal verification is introduced, through reasonable setting of constraints, adding cutpoints and other means, the efficiency of fault coverage calculation has been significantly improved.

Acknowledgments
This research was financially supported by the science and technology project of State Grid Corporation of China. The project name is, Key Technology and Application Research for Security MCU in Smart Endpoint Device. The project No. is 54681618009.

References
[1] Bellotti M, Mariani R, 2010 How future automotive functional safety requirements will impact microprocessors design vol 50 (Microelectronics Reliability) pp 1320-1326.
[2] IEC61508. 2010 Functional safety of electrical/electronic/programmable electronic safety-related systems (International Electrotechnical Commission, IEC).
[3] Asif J, etc, 2018 Proving the capability of Arm IP for functional safety application (2018 19th International Workshop on Microprocessor and SOC Test, Security and Verification (MTV), Austin, TX, USA).
[4] 2010 Z01X Simulator safety verification user guide vol 3 (Synopsis).
[5] Erik S, Tom S, Achutha K M V, 2015 Formal verification an essential toolkit for modern vlsi design (Elsevier Inc).
[6] Marchese S, 2017 When correct is not enough: formal verification of fault-tolerant hardware (OneSpin Whitepaper).
[7] 2020 VC Formal verification user guide, version Q vol 3 (Synopsis).