Quality-by-Design Approach to Developing Servers

● Yasunori Fujioka ● Mutsuhiro Tanaka

(Manuscript received July 10, 2007)

Given the shorter product life cycles in recent years, the ability to promptly develop and produce high-reliability products is now the key to success in business. Fujitsu has significantly reduced the development span of its core products (servers). As a result, releasing products within a fixed time limit has become increasingly difficult using traditional development methods. Moreover, any problem that may occur at the evaluation or mass-production stage has a critical impact on product assurance. Under these circumstances, we have been working on improving product assurance based on the "quality-by-design" approach. This activity helps us look at problems involving LSI remakes necessitated by design errors as one of the major factors causing scheduling delays at the development stage. This paper describes Fujitsu's activities for improving server product assurance to achieve zero LSI remakes relative to proprietary LSI circuits for servers, and discusses the simulation techniques involved.

1. Introduction

The verification work in hardware development generally consists of four main phases: design review, simulation (SIML), design verification test, and validation test, as shown in **Figure 1**. Manufactured products are actually used for design verification test and validation test. Any hardware design change made in either of these verification phases costs a great deal of money for LSI correction. It also takes much time to redesign, re-verify, and reproduce the LSI upon return due to failure in evaluation, which adversely affects shipment time. Various improvements have been made to design methods for assuring the design quality of servers before the LSI manufacturing process. These include simplifying formal verifications and third-party reviews using hardware description language (HDL), and checking actual equipment operations using high-speed, large-scale SIML equipment.

Due to these improvements, design failures that may occur in actual equipment verification have declined year by year, except for LSI correction. There are still problems to solve regarding timing and configuration dependence prior to the actual equipment verification process, particularly for the memory-controlling LSI. Accordingly, the critical point is identifying problems in the memory-controlling LSI before the LSI manufacturing process in order to assure product quality, cost, and delivery.

Given these circumstances, this paper reviews the quality assurance of hardware products being newly developed from the standpoints of product assurance and actual equipment evaluation. The basic items to be reviewed entail:

"Why is the quality-by-design approach implemented?"

"Why are problems left undetected?"

"Where should we detect the problems?"

"How should we identify the problems?"

2. Issues in quality assurance

The memory-controlling LSI mainly provides a traffic control function to quickly process requests from multiple CPUs and I/O processors. It has a very complex and large-scale logical configuration to support this function. Therefore, it must be developed by a number of engineers, often resulting in a misuse of interface specifications, ungenerated conditional expressions, and other mistakes. Moreover, the waterfall model in which a specification study, detailed design, and logic entry are conducted sequentially in a long-term flow is adopted for development. Thus, a problem detected after logic entry may resurface at more than one location in the development process.

Timing errors may also occur due to the separation of work for Logical-block SIML and component LSI SIML.

Patterns should basically be checked in Logical-block SIML, but three-dimensional complex patterns consisting of internal state, external state, and input patterns are physically and temporally difficult to check. Drawings and specifications are used to check these patterns.

The current high-speed, large-scale SIMLs are intended to identify the problems above so that no failure occurs in the actual equipment. However, such SIML takes much more time than the actual equipment. One minute operation for the actual equipment corresponds to 14 days of operation for SIML. In terms of speed, using SIML to check operation is not practical given the very complex logic of an LSI and the time (more than an hour) needed for the actual equipment. In light of these facts, major possible issues include:

1) Developing an accelerated SIML that includes actual equipment operations as a technique to detect problems
2) Preventing bugs from increasing via quick feedback to developers to improve quality-by-design

3. Existing SIML

SIML has the same environment configura-

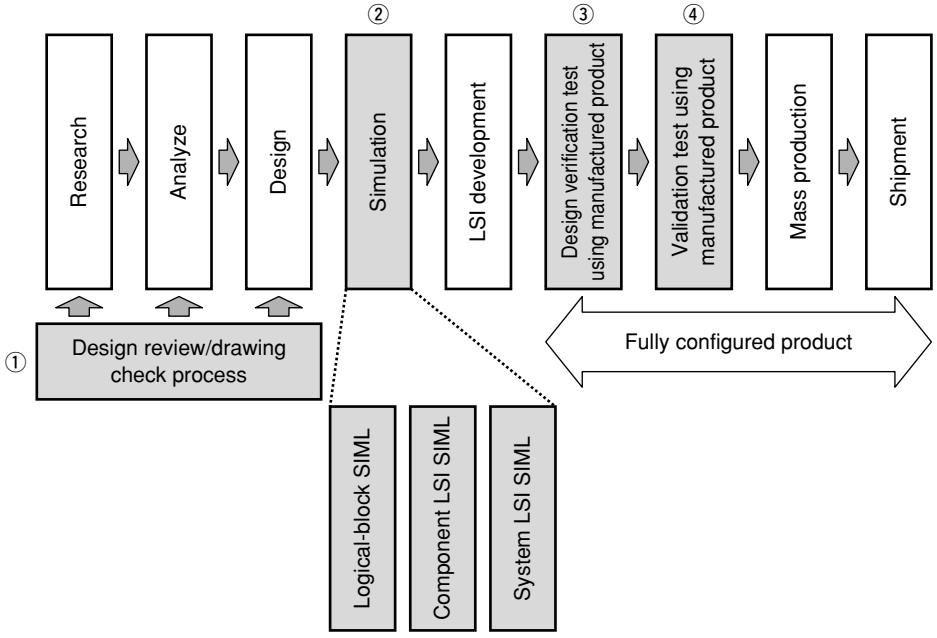


Figure 1
Flow of developing general hardware products.

tion as the actual equipment in a simulator as shown in **Figure 2**, and is equipped with CPUs, I/O processors, a memory unit, and the target memory-controlling LSI, with each unit using HDL.

In preparation, the test program to be used in the actual equipment is preloaded in the memory unit. Then a start trigger is applied to the test program to start simulation verification.

The test program is coded using instruction code (e.g., load, store, operation instructions) compliant with the hardware architecture in the same way as the OS. To actually test the memory-controlling LSI, the following problems should be solved:

1) CPU cache wall

When the test program issues instruction fetch or operand access to apply load to the memory-controlling LSI, the CPU cache memory processes the subsequent access operations from memory via the memory-controlling LSI. To access the memory-controlling LSI, the CPU cache function must be invalidated. This involves executing multiple instructions, however, and makes it difficult to increase access load on the memory-controlling LSI.

2) Overhead caused by instruction fetch

The CPU hardware issues instruction fetch to memory as needed because CPU operation requires instructions. Fetch processing keeps other accesses waiting or synchronizes the access order, thus making it difficult to increase load on the memory-controlling LSI and create a wide variety of access patterns.

3) Overhead involved in the test program flow

A test generally consists of environment setting (setup), execution (test), and decision (check). Because the test program performs environment setting and decision, the memory-controlling LSI remains nearly idle.

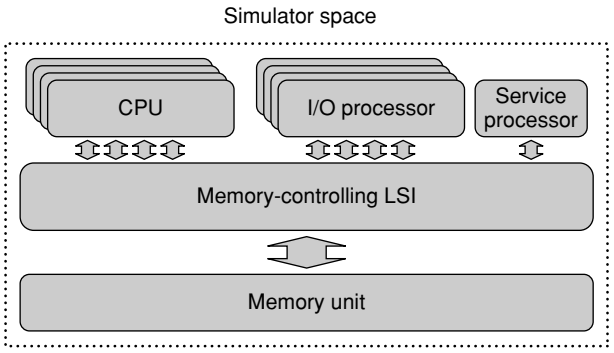


Figure 2
Summary of simulation.

4. Developing dedicated LSI tester

To solve the SIML problems described in the previous section, a dedicated LSI tester has been developed (**Figure 3**). This tester achieves a very attentive evaluation that enables direct access to the memory-controlling LSI by removing the CPU cache wall and overheads.

As the control method, the LSI testers are initially allocated to each interface of the memory-controlling LSI for the CPUs, I/O processors, and service processor. Then test cards describing the access procedure are added for each LSI tester to ensure independent operation of the LSI testers. This method helps remove the "CPU cache wall". The test cards also eliminate the need for instruction fetch operation in executing a test so that instruction fetch is used as one of the test categories.

The LSI tester uses four major techniques to avoid the overhead involved in the test flow, and thus enables diverse access operations and timing patterns.

While common testers are considered to merely check functions, the LSI tester is intended to generate various timing patterns while running because it includes actual equipment operation.

4.1 Integration between test program and hardware logic

Typical test programs often include trap points to check for conflicting operations by comparing data or taking other actions.

Hardware logic locally uses a logical conflict checker as the RAS function. The LSI tester has a logical conflict checker that operates the same way as normal hardware logic and uses the trap concept of the test program to perform check operations associated with the test card. This mechanism allows us to check for regular interface conflicts and data failures, as well as specific timing patterns. Should the LSI tester receive an unexpected response from the memory-controlling LSI, it basically regards the response as an error and outputs an error message, unlike normal logic that ignores it. Traps are cyclopaedically set based on restrictions on the interface specifications.

4.2 Mechanism to change timing

For a circuit with relatively small logic, all timing variations can be generated in a test. Conversely, for a circuit with very complex logical conditions, the key is how many systematic timing patterns can be generated, along with the combined use of Logical-block SIML.

To generate as many timing patterns as possible, the LSI tester has a dam in the output interface to the memory-controlling LSI for providing a waver control function as shown in Figure 3. This function can be used to fluctuate the load. For example, it can output accumulated requests in one burst, optionally extend or reduce the request intervals, and divide and mix multiple requests as required. The dam can also respond to the memory-controlling LSI more rapidly than the actual equipment.

4.3 Simple logic structure

Because the LSI tester operates based on hardware logic just like the memory-controlling LSI, it may generate bugs. To reduce the number of logical bugs in the LSI tester to lower than that in the memory-controlling LSI, the tester employs very simple hardware logic. It uses an instruction code with which to pass instructions directly to the memory-controlling LSI, instead of using hardware architecture instruction code. Decoding and emulation logic, in particular, have been deleted. The tester also performs a data-pattern coverage tradeoff.

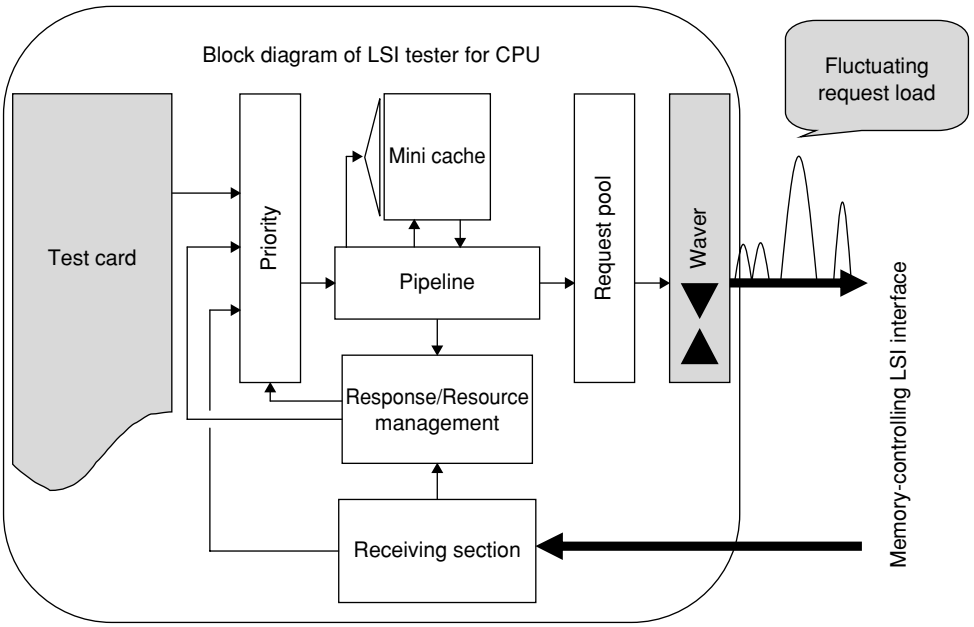


Figure 3
Outline of LSI tester.

Accordingly, the logical gate scale of the LSI tester is so compact that many LSI testers can be installed in a limited simulator space. The LSI testers can be allocated to all interfaces of the memory-controlling LSI, which allows SIML to generate timing variations in the maximum configuration.

4.4 Data consistency in competing blocks

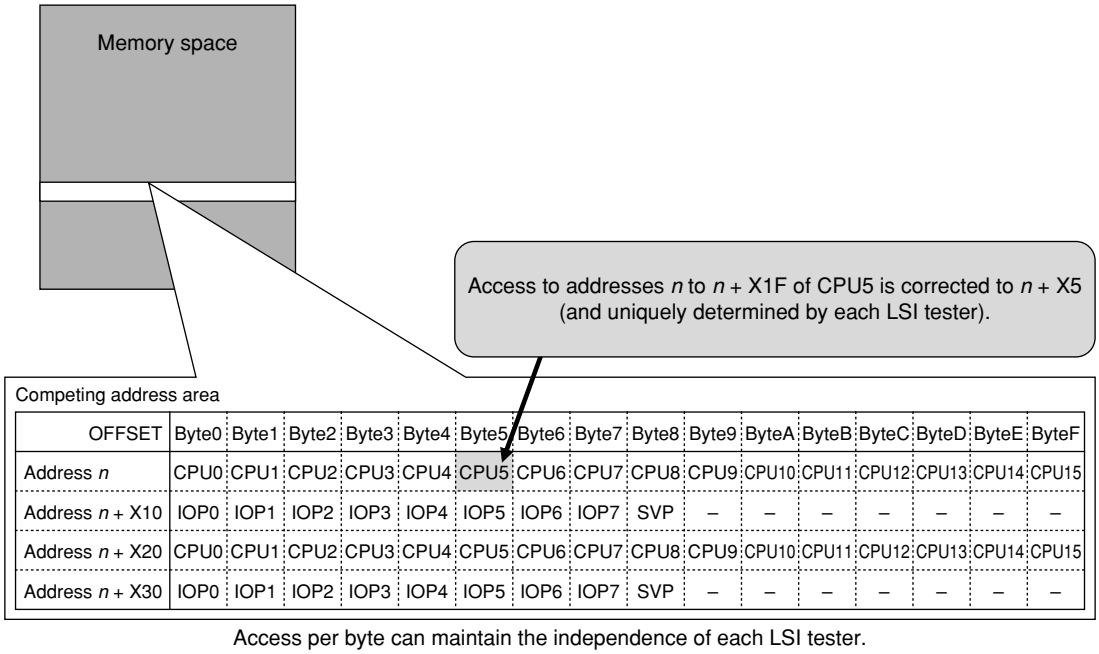
When multiple testers in a system competitively use the same address to check for data conflicts in the test, a talking interface must be configured between the testers to control data update timing. In this case, talking imposes an overhead, resulting in the synchronization of testers in a large-scale configuration and concerns about less-varied timing patterns.

As an improvement plan, a competing address area is assigned to each tester per byte as shown in **Figure 4**. In this area, one block consists of 64 bytes. Each tester can only update or reference the byte offsets assigned to the tester. The testers maintain independence in checking for data conflicts without being synchronized with other testers, even in identical address competition.

5. Results of SIML using LSI tester

Memory-controlling LSI SIML was conducted using the techniques described in the previous section. As a result, the SIML detection rate for overall failures was dramatically improved compared with existing models. In particular, the problem of timing failure caused by prolonged running in the actual equipment tests was drastically reduced. This result demonstrated that enhanced timing variation in SIML had worked.

Some failures that may be detected in actual equipment tests still occurred, however. The RAS function could not be extracted satisfactorily in SIML. The clock control and test functions were also difficult to improve using only the evaluation techniques that we employed this time. We have therefore been working to enhance the evaluation techniques for such hard-to-improve functions.



OFFSET	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8	Byte9	ByteA	ByteB	ByteC	ByteD	ByteE	ByteF
Address *n*	CPU0	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6	CPU7	CPU8	CPU9	CPU10	CPU11	CPU12	CPU13	CPU14	CPU15
Address *n* + X10	IOP0	IOP1	IOP2	IOP3	IOP4	IOP5	IOP6	IOP7	SVP	–	–	–	–	–	–	–
Address *n* + X20	CPU0	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6	CPU7	CPU8	CPU9	CPU10	CPU11	CPU12	CPU13	CPU14	CPU15
Address *n* + X30	IOP0	IOP1	IOP2	IOP3	IOP4	IOP5	IOP6	IOP7	SVP	–	–	–	–	–	–	–

Access per byte can maintain the independence of each LSI tester.

Figure 4
Data consistency.

6. Future issues

The SIML evaluation techniques have been improved on a daily basis along with the innovations made in SIML technology, including the measures implemented this time. However, our major future issue is to identify and target the points that should be evaluated; that is, to develop a technique to extract evaluation points. Current techniques for the extraction of LSI logic evaluation include formal verification, V-shaped evaluation with the waterfall model, and matrix review using the points listed from previous experience. Although the technique of the completeness test with the route between conditional judgments may be used in software development, this is not possible in hardware development. Because hardware involves many conditional judgments and most logical modules always operate through linkage with other modules, the variations in timing patterns to be extracted will become astronomical figures.

To prevent bugs from increasing, the round-trip development model is more effective than the existing waterfall V-shaped model where tests are executed for finished products. In the round-trip model, production and testing alternate in small units with results accumulated in the logic entry phase, thus stabilizing the quality of each minimum component. Building a test environment at the design stage and making adjustments with other processes such as LSI implementation are the keys to making the most of the round-trip model. These key points should inevitably be developed. Since implementation of the round-trip model affects the overall development process, we will initiate studies toward its realization (**Figure 5**).

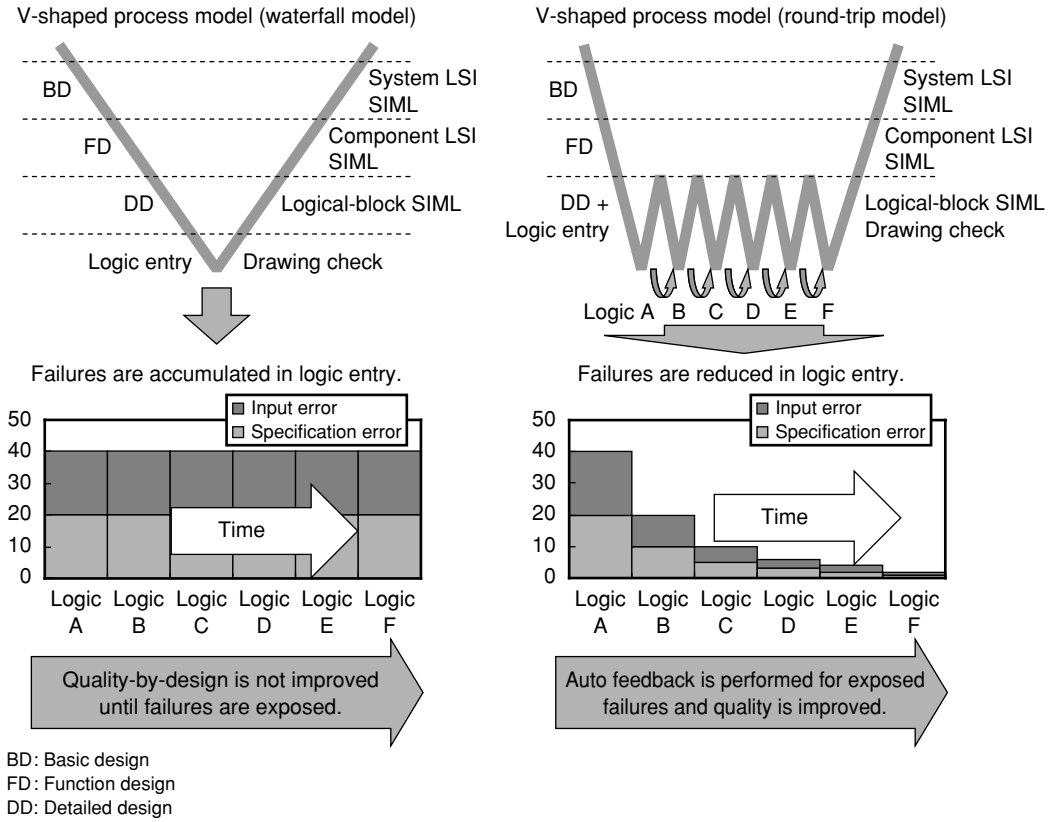


Figure 5
V-shaped process model.

7. Conclusion

This paper described our activities for assuring the quality of server hardware products. As a new challenge of having the quality assurance unit participate in development, we encouraged participation of that unit in design verification tests.

Under ordinary circumstances, the development unit is responsible for evaluation. However, due to recent reductions in the span of development or other reasons, front-loading evaluation at the development stage has replaced actual equipment evaluation. In the future, the development span will be made even shorter, and our activities for quality assurance will prove more significant.

Yasunori Fujioka, *Fujitsu Ltd.*
Mr. Fujioka graduated from Mitoyo Technical High School in Kagawa Prefecture in 1968. He joined Fujitsu Ltd., Kawasaki, Japan in 1970, where he has been engaged in quality assurance activities for Fujitsu mainframe servers, UNIX servers, and storage system products.

Mutsuhiro Tanaka, *Fujitsu Ltd.*
Mr. Tanaka graduated from Shimizu Technical High School in Shizuoka Prefecture in 1983. He joined Fujitsu Ltd., Kawasaki, Japan in 1984, where he has been engaged in mass production and quality assurance activities for Fujitsu mainframe server products.