DOI : 10.3745/JIPS.2008.4.4.121

Tester Structure Expression Language and Its Application to the Environment for VLSI Tester Program Development

Masayuki Sato*, *, Hiroki Wakamatsu**, Masayuki Arai***, Kenichi Ichino***, Kazuhiko Iwasaki*** and Takeshi Asakawa******

Abstract: VLSI chips have been tested using various automatic test equipment (ATE). Although each ATE has a similar structure, the language for ATE is proprietary and it is not easy to convert a test program for use among different ATE vendors. To address this difficulty we propose a tester structure expression language, a tester language with a novel format. The developed language is called the general tester language (GTL). Developing an interpreter for each tester, the GTL program can be directly applied to the ATE without conversion. It is also possible to select a cost-effective ATE from the test program, because the program expresses the required ATE resources, such as pin counts, measurement accuracy, and memory capacity. We describe the prototype environment for the GTL and the tester selection tool. The software size of the prototype is approximately 27,800 steps and 15 man-months were required. Using the tester selection tool, the number of man-hours required in order to select an ATE could be reduced to 1/10. A GTL program was successfully executed on actual ATE.

Keywords: *VLST test, VLSI tester, ATE, tester language, GTL, Tester selection tool*

1. Introduction

VLSI chips have been tested by general-purpose automatic test equipment (ATE), which interprets a test program in the main memory and controls test resources such as AD/DA converters and power supplies. Although each tester has a similar structure, test languages differ from vendor to vendor. In other words, there is no inter-compatibility among ATE software. This is one reason why the test cost of VLSI chips has not decreased, while the cost-performance of ATE has improved. Since the test cost is viewed as a problem for a large-scale SoC (system on a chip) [1], it is important to improve the tester language.

The architecture of VLSI testers has evolved as the device under test (DUT) has evolved. During the 1970's, ATE was based on the shared-resource architecture, where tester resources are used by multiple pins in the time division method [2,3]. The per-pin tester was then developed, where full tester resources are assigned to each pin [4]. Recently, the test-processor-per-pin tester and the time driven tester have been developed [5]. Testers that focus on design for testability (DFT) have also been introduced to the market [6,7]. A board-mounted tester for a dedicated chip was also developed [8].

The software of the tester is important in testing VLSI chips. As tester architecture progresses, the languages used to describe test programs have evolved in the same manner as programming languages, as shown in Figure 1. The assembly language was used on testers in the 1970's [9]. Then, in the 1980's, FORTRAN-like [10] and BASIC-like languages were developed and have become widely used. A PASCAL-based language was also developed, but the language has not yet been used. A C-like language was developed in the 1990's, and a GUI-based language has recently been made available.

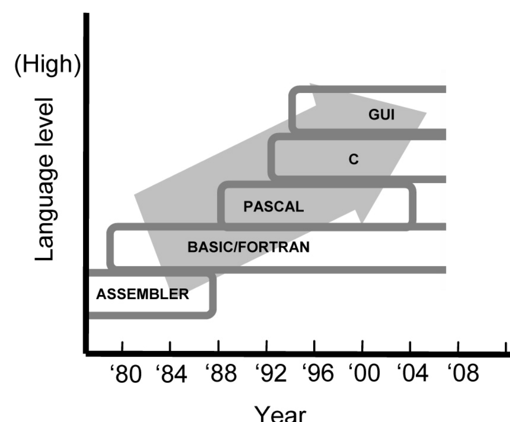

Fig. 1. Evolution of tester language: Assembler-like language to GUI-based language.

Manuscript received 10 September, 2008; accepted 27 September, 2008.

Corresponding Author: Kazuhiko Iwasaki
* Genesis Technology Inc., presently Taiyo Yuden Corp. (m-satou@jty.yuden.co.jp)
** Genesis Technology Inc., presently Oki Engineering Co. (wakamatsu447@oki.com)
*** Tokyo Metropolitan University (m-arai@tmu.ac.jp, ichino@coreappli.jp, iwasaki@eei.metro-u.ac.jp)
**** Tokai University (asakawa@tokai.ac.jp)

Copyright ⓒ 2008 KIPS (ISSN 1976-913X)

To develop a test program, an engineer must be familiar with not only the test language, but also the ATE structure, ATE operations, and the DUT test structure. Therefore, debugging a test program requires an actual ATE environment, resulting in a costly design process. To alleviate this problem, a software tool called a virtual tester is also introduced [11]. The virtual tester simulates a VLSI tester and can be used to verify and debug software on the tester.

There have been efforts to standardize the test description language, such as the standard test interface language (STIL) [12]. The STIL satisfies the requirement for the common format of test patterns, and thus has become widely applied. However, further efforts at standardization are required in order to describe the tester program, because the STIL does not necessary take the tester architectures into account, making it difficult to set the parameters to the test resources. In addition, many testers have yet to support STIL as a native language. STIL must be converted to the original language of the tester before test application, reducing the productivity and portability of the program.

VLSI developers, including fabless companies, must develop a test program for each device and piece of ATE used. The debugging process begins directly after the first-silicon. As described above, developing/debugging the test program requires significant manpower. This is one disadvantage to shorten the time-to-market (TTM) and time-to-volume (TTV). In other words, prompt test program development leads to shorter VLSI debugging processes and feedback to yield improvement.

In order to overcome these difficulties, a new tester language must be developed. In the present paper, we propose a tester structure expression language, namely, the general tester language (GTL). The GTL is an attempt to improve the readability and portability of the tester software. In addition, by developing the interpreter of the proposed language for each target tester, the tester can execute the tester program without conversion. Based on the proposed language, we have developed the prototype tool for programming and selection of an applicable tester. The developed GTL program was successfully interpreted on an actual ATE, and the test program was executed on actual VLSI testers.

The remainder of the present paper is organized as follows. In Section 2, we report the typical tester structure and typical test program statements, indicating the difficulties in the conventional environment. In Section 3, we present the concept and an example of the proposed tester language, GTL. Section 4 shows the architecture of the prototype tool developed under the proposed language, including experimental results. A GTL program executed on actual VLSI testers is reported in Section 5. Section 6 summarizes the present paper.

2. Tester Structure and Tester Language

2.1 Tester Structure

Figure 2 shows the logical architecture of the general-purpose tester. The test program is stored in the main memory. The central processing unit (CPU) executes the test program by interpreting the test program and controlling the measurement instruments, which are referred to as tester resources. Thus, general-purpose testers can be regarded as a control system with many tester resources and can be logically expressed as a bus-based architecture.

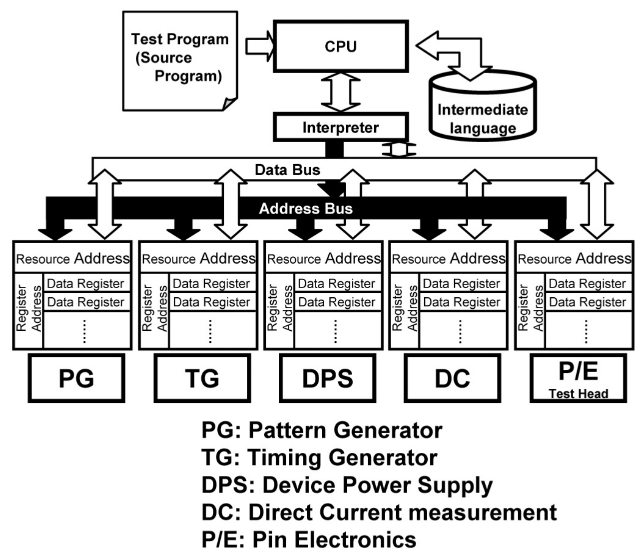

Fig. 2. Logical structure of typical automatic test equipment (ATE).

A tester has many tester resources, as described in the following:

(1) Pattern Generator (PG)

This unit stores the logic value of the test vectors and generates the test pattern.

(2) Timing Generator (TG)

This unit generates the timing for the test vector input to the DUT. It also generates comparison timing for the test responses output from the DUT. The unit is indispensable for AC measurements of the device.

(3) programmable Device Power Supply (DPS)

This unit supplies the voltage to the DUT.

(4) DC measurement (DC)

This unit measures the DC characteristics of the DUT.

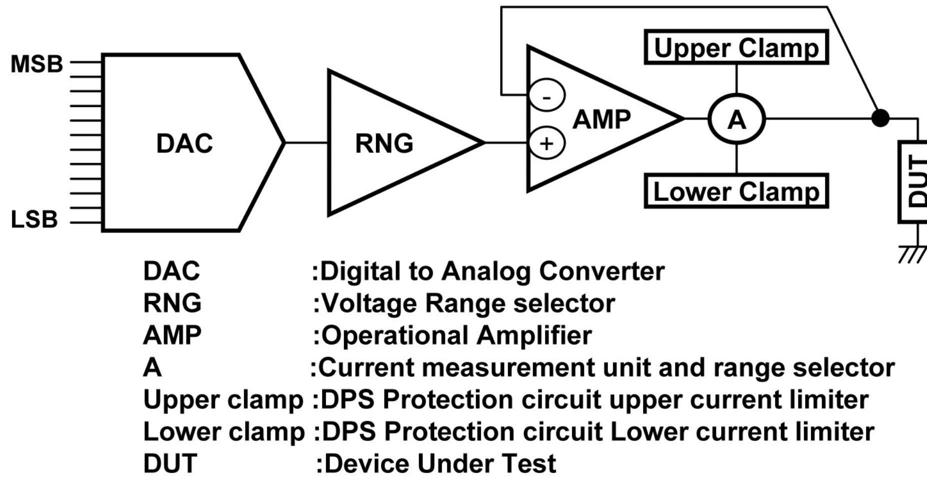

Fig. 3. Structure of the programmable device power supply (DPS).

(5) Pin Electronics (P/E)

This unit consists of drivers, which input signals to the DUT, and comparators, which compare the output from the DUT. Pin electronics are designed to work correctly for wires having lengths of up to 1 meter. For most ATE, the impedance of P/E is 50 ohms.

Each tester resource has its own resource address. When the CPU in the ATE sets a parameter for a tester resource, it specifies the memory address assigned to the register for the parameters and sets the data on the Data Bus.

As an example of tester resources, Figure 3 shows the DPS circuit consisting of the following: Digital to Analog Converter (DAC), voltage Range selector (RNG), operational Amplifier (AMP), current measurement unit and range selector (A), and upper and lower clamp circuits. For many ATE components, the DAC has 12-bit resolution and generates voltages ranging between +2.048 V and -2.047 V. The range of the RNG circuit is selected to be 0.8 V, 8 V, and 80 V. The AMP supplies the current to the DUT by observing the sense line (S) and controlling the force line (F). The current measurement range can be selected from among 800 nA, 8 μA, 80 μA, 800 μA, 8 mA, 80 mA, and 800 mA. These values are set to the designated register in the DPS via the Data Bus.

2.2 Survey of Tester Languages

The first author of this manuscript was a member of Working Group WG2 (Test Engineering) of the Semiconductor Technology Roadmap of Japan (STRJ) and initiated a survey on the tester languages [13,14]. The WG2 requested the cooperation of Japanese tester vendors, and focused on five testers: Testers 1 and 2 were manufactured by A Corp., Testers 3 and 4 were manufactured by B Corp., and Tester 5 was manufactured by C Corp.

The sizes of the tester programs were evaluated for the five testers mentioned above. The target device was a standard edge-trigger flip-flop, 74HC74. A test program was designed for each tester, and the testers were compared in terms of the number of code lines, as shown in Table I. The types of language and the number of program lines are as follows: Tester 1 (FORTLAN-like) = approximately 400 lines, Tester 2 (C-like) = approximately 760 lines, Tester 3 (BASIC-like) = approximately 200 lines, Tester 4 (C-like) = 360 lines, and Tester 5 (C-like) = 240 lines. Since Tester 3 adopts a BASIC-like language, this tester had the smallest number of lines. In the following subsection, the description of the DPS for the circuit will be shown.

Table 1. Tester program sizes for edge-trigger flip-flop 74HC74 for five types of ATE.

ATE maker	A Corp.		B Corp.		C Corp.
tester	Tester 1	Tester 2	Tester 3	Tester 4	Tester 5
language	FORTRAN-like	C-like	BASIC-like	C-like	C-like
number of program lines	400	760	200	360	240

Here, we discuss in detail the differenced and similarities among the tester languages. In each program, for the DPS, we set the input voltage as 0 V, the voltage range as 8 V, the measurement current range as 0.8 A, and the clamps as ±400 mA. The tester programs are listed below according to the year that each tester was developed. First a part of the program for Tester 1 (FORTRAN-like) is presented, followed by a part of the program for Tester 3 (BASIC-like) and Tester 2 (C-like). Due to space limitations, the programs for Testers 4 and 5 are presented in Appendix A. As a result of the limitation of the parameter range, accurate setting with our requirement was

impossible for Testers 4 and 5.

(Tester 1)
VS1 = 0.000 V, R8V, M(0.8 A), 400MA, -400MA

This tester was developed in 1980 and is now widely used. A FORTRAN-like language is used. The term VS1 indicates the name of the resource. According to the parameters on the right-hand side, the electrical characteristics for the resource are determined. The applied voltage is set to be 0.000 V, the voltage range is set to be 8.0 V by R8V. The current range is set to be 0.8 A by M(0.8 A), and the upper/lower clamp current is set to be 400 m/-400 mA. The order of the parameters is fixed, and capital characters must be used.

(Tester 3)
DEFINE section: VCC = BS1(4R,4C)
DATA section: VCC = 0.0 V

Tester 3 became commercially available in 1990 and is also widely used at present. It uses a BASIC-like language. The DEFINE division describes the resources used, while the DATA division describes the parameters to be set for the resource. The voltage supply is represented by VCC, and BS1 is the designation of the voltage source. Parameter 4R indicates that the voltage range is 2.0 V, and 4C means that the current measurement range is 200 mA. These expressions are taken from the description for the early Fairchild testers. The expression VCC = 0.0 V indicates that the voltage applied to BS1 is 0.0 V. The values in the DATA division are transferred to the ATE memory by a direct memory access (DMA) mechanism.

(Tester 2)

```
DPSVSIM    dpsvsim;
dpsvsim.pin(VS1);
dpsvsim.SRng(R8V);
dpsvsim.SVal(0V);
dpsvsim.MRng(M800MA);
dpsvsim.CPVal(400mA);
dpsvsim.CMVal(-400mA);
dpsvsim.Load();
```

Tester 2 was developed in 1999 and uses a C-like language. The order of the parameter setting can be changed. The first and second lines define the voltage source, designated VS1. The third line, SRng(R8V), indicates that the voltage range is 8.0 V. The fourth line, Sval(0V), indicates that the applied voltage is 0.0 V. The fifth line, MRng(M800MA), indicates that the current measurement range is 800 mA. The sixth and seventh lines are used to set the clamp current to 400/-400 mA, respectively. The final line sends the parameters to the ATE resource registers via the Data Bus.

2.3 Description of the Test Data

In this subsection, we examine the test data format. For all testers, the logical states of test patterns were identical: 0 for input low, 1 for input high, L for output low, H for output high, and X for the ‘don’t care’ value. The following is an example of the test pattern for Tester 1:

```
LPAT PFCT2
CHANNEL 1-8,10-13
CFPF  NOP/T1        !1000XX1000XX
      NOP/T2        !0001LH0001LH
                :
STPS        /T11            !1001XX1001XX
END
```

The first line indicates that what follows is a test pattern designated PFCT2. The second line specifies the pin number for the test. The test pattern is described between CFPF and STPS statements. ‘END’ indicates the end of the description.

2.4 Program/Data conversion

For a larger VLSI, such as a SoC, a test program becomes larger and more complicated due to increasing pin-counts and larger integration. The test program is used not only by the test engineer but also by the process line managers, operators, and defect analysis engineers. Therefore, it is advisable to design test programs that are more structured and readable. As shown in the previous subsections, the commercially available tester language lacks readability and portability.

It is quite difficult to convert a test program to/from a test program for a different ATE vendor. This is because the user must be familiar with the details of both ATE components, which often have different ranges. Figure 4 shows a possible method by which a test program is converted to a test program on a different ATE. First, a test program is developed on a specific VLSI tester according to the constraints of the ATE (Fig. 4(a)) and is then converted to the other (Fig. 4(b)) [15]. The drawback of this technique is as follows. The number of combinations increases rapidly as the number of ATE components increases. To ease this difficulty, a database-based conversion technique has been developed, where the resources for each tester is stored in a database, such as an EXCEL file, as shown in Fig. 4(c).

From the above experiments, the following insights were obtained:

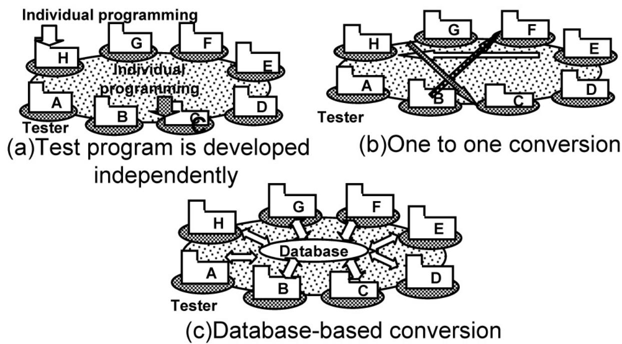

Fig. 4. Test program conversion methods: (a) Independent development, (b) One-to-one conversion, and (c) Database-based conversion.

(1) For some languages, engineers must know the order of parameter settings as well as the parameter itself, resulting in difficult program development and conversion.

(2) C-like languages tend to become longer because of individual parameter settings.

(3) Some languages supply multi-parameter statements, which can reduce the number of code lines, but the format is vendor-dependent.

(4) The test pattern format is approximately the same for each tester. This indicates that the pattern can be converted much easier than a tester program.

3. Tester Structure Expression Language

To alleviate the difficulties described in the previous section, we propose a novel language for VLSI testers, namely, the general tester language (GTL). The features of the GTL are as follows:

(1) The GTL adopts a C-like format. The parameters of tester resources are set based on the arguments of functions that are defined according to the logical structure of the testers.

(2) Test patterns are described in the conventional manner, that is, in the 0, 1, H, L, X format.

(3) The GTL is easy to develop and understand for not only test engineers, but also process line managers, operators, and defect analysis engineers.

As described in a previous section, ATE has similar resources and features, even if the vendors are different. It is possible to describe a VLSI tester using software, i.e., a set of functions. Figure 5 shows an example of this concept.

As examples, the functions for the PG, DPS, Pin, and DC are defined below:

(PG)

PATTER Pattern name
CHANNEL(Pin)
NOP(Timing set number, Pattern Description)
…
STPS(Timing set number, Pattern Description)
END

(DPS)

VS(Unit number, Voltage, Voltage Range, Measurement Current Range, Upper Current Clamp, Lower Current Clamp)

(Pin)

Pin(Pin number, input Hi level, input Low level, Terminal Voltage, Wave Mode, CLK number, Driver/Comparator/IO, Output Hi level, Output Low level, Strobe)

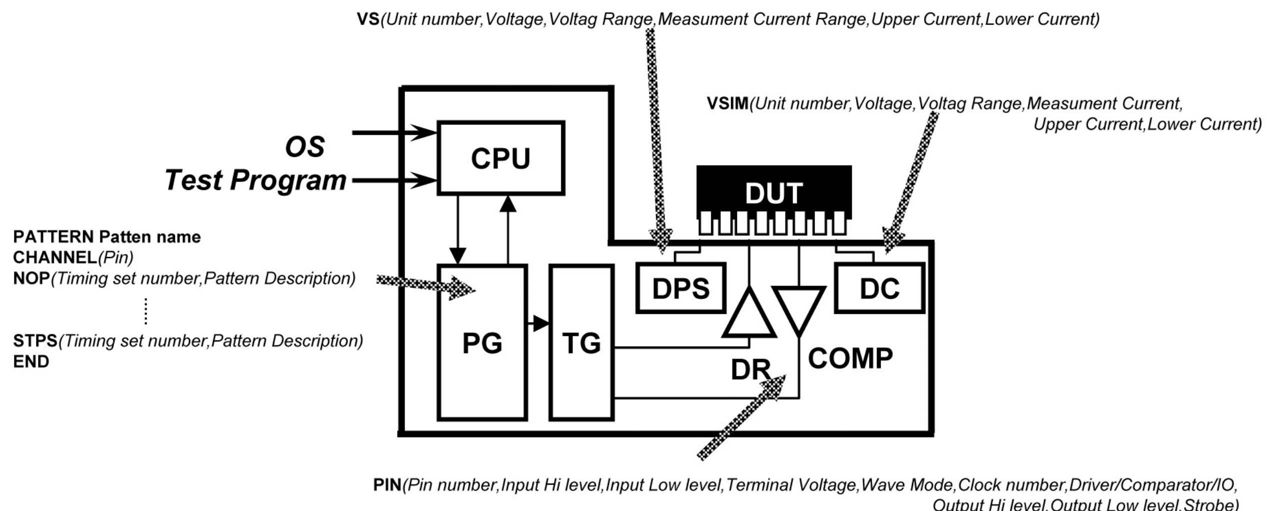

Fig. 5. Tester resources and their description by functions.

(DC)

VSIM(Unit number, Voltage, Voltage Range, Measurement Current Range, Upper Current Range, Lower Current Range)

For example, consider the function for the DPS. The term VS, which stands for voltage source, is the designation of the function. The parameters are described as arguments of the function. The function VS() has the following six arguments: unit number, voltage applied, voltage range, measurement current range, and upper/lower clamp current. These arguments sufficiently cover the parameters necessary to set the five testers mentioned earlier. A typical expression for this resource is as follows:

VS(1, 0 V, 800ma, 400ma, -400ma)

If engineers are not concerned with or do not wish to specify some of the parameters, the following expression is also acceptable:

VS(1, 1V, , , ,)

All of the resources used to describe ATE are surveyed with the cooperation of the WG2 in STRJ. The results, including the device power supply, DC descriptions, voltage sequences, measurement conditions, and test pattern description, are shown in Table 2. For additional details, please refer to [13].

Using the functions listed in Table II, each test program on each ATE can be described by a set of statements. The logical structure of the program can be shown as a tree structure, as shown in Figure 6. The proposed language represents the structure of the testers. That is, each function represents one tester resource. In other words, each function corresponds to a resource address on the tester bus shown in Figure 1. The order of parameters is based on their sub-addresses.

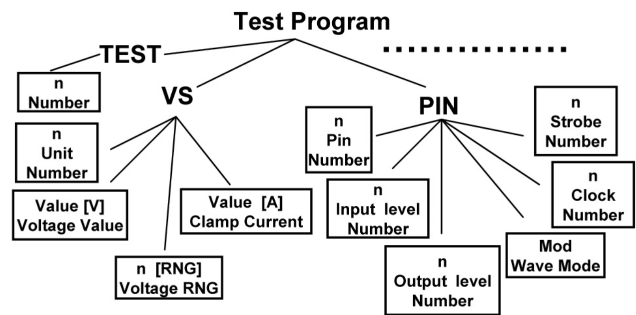

Fig. 6. Test program structure for the general tester language (GTL).

Table 2. Function for the general tester language (GTL).

Tester Structure Expression Language		
C Language Function Definition Description		
Tester Description	Device Power Supply	**VS**(*Unit number,Voltage,Voltage Range,Measurement Current Range, Upper Current Clamp ,Lower Current Clamp)*
	DC Test: Voltage Source Current measure	**VSIM**(*Voltage,Voltage Range,Measurement Current Range, Upper Current Clamp ,Lower Current Clamp)*
	DC Test : Current Source Voltage measure	**ISVM**(*Current,Current Range, Voltage Measurement Range, Upper Voltage Clamp ,Lower Voltage Clamp)*
	Pin description	**Pin**(*pin number,Input Hi level,Input Low level,Termination Voltage,Wave Mode, Clock Driver/Comparator/IO, Output Hi level,Output Low level,Strobe)*
	Voltage sequence	**TIME**(*Sequence number,Wait Time,Unit name)* **SRON** **SROF**
	Test number	**TEST**(*Test number)*
	Measurement Command	**MEAS**(*Measurement name)* **REG**(*Unit name,Register name,Data)* **SEND**(*unit name,data)*
	Wait time	**WAIT**(*Time)*
	Test program Start	**MAIN**(*program name*) {
	Test program finish	}
	Test Stop	**STOP**
	Brunch Command	**if**(*ARG1(5)<>0)*
	Loop Control Command	**for** *(I=0; I<10; I++)*
	Limit Description	**LIMIT**(*Measurement Unit,Upper Limit,Lower Limit)*
	Pin List	**PINLIST**(*Pin List name,Pin number)*
	Test Rate	**RATE**(*Rate time)*
	Timing Description	**CLK**(*Clock number,ACLOK Value,BCLK Value,CCLKValue)* **STRB**(*Strobe number,Strobe value)*
	Pattern Description Pattern description end	**PATTER**(*Pattern name)* **CHANNEL**(*Pin description)* **NOP**(*Timing set number,Pattern description)* ∫ **STPS** *(Timing set number,Pattern description)* **END**

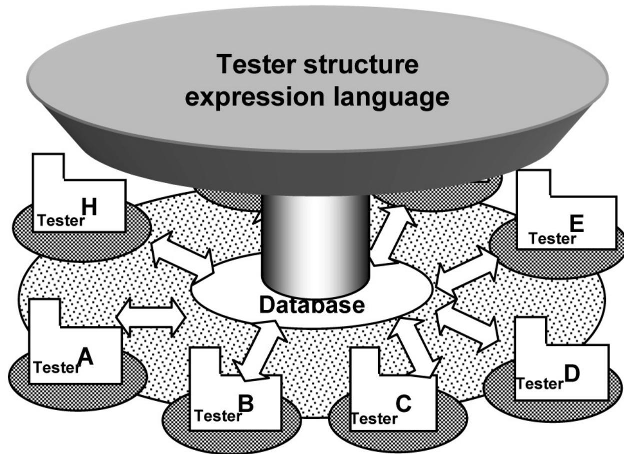

Fig. 7. Program conversion from the GTL program.

By implementing an interpreter of the GTL functions for the native tester language, test execution can be performed without converting the proprietary test program. If the native language of the tester is C-like, it can be implemented simply by the definition of the GTL functions. Even if the native language is not C-like, function calls can be implemented by multi-parameter macros, which are supported by many current testers.

The proposed language has no constraints on the range of each parameter, so that the proposed language is easily convertible to various testers. It is also possible not to set some of the parameters. The ranges of the parameters that are dependent on each tester can be checked by tools such as the tester navigator mentioned in the following section.

4. Application of the GTL to the Test Solution Environment

4.1 Prototype Structure

We developed a prototype tester selection and programming environment with the tester structure expression language, GTL. The prototype tool provides an environment consisting of the following components, as shown in Figure 8:

(1) Interactive editor
(2) Tester resource evaluator
(3) Tester inquiry
(4) Tester navigator
(5) Test pattern converter (third-party product)
(6) Virtual tester (third-party product)
(7) Test specification sheet

The user can use the tool via a Web-based interface during program development. The pattern converter and the virtual tester are developed by third vendors. Each component will be described in the following.

4.2 Software Components

(1) Interactive editor

The interactive editor is a screen editor specialized to efficiently input a GTL program. When the name of a function is input, the interactive editor displays a list of the parameters required for the function, prompting the parameters to be input one by one. Thus, the user can develop tester programs even if without exact knowledge

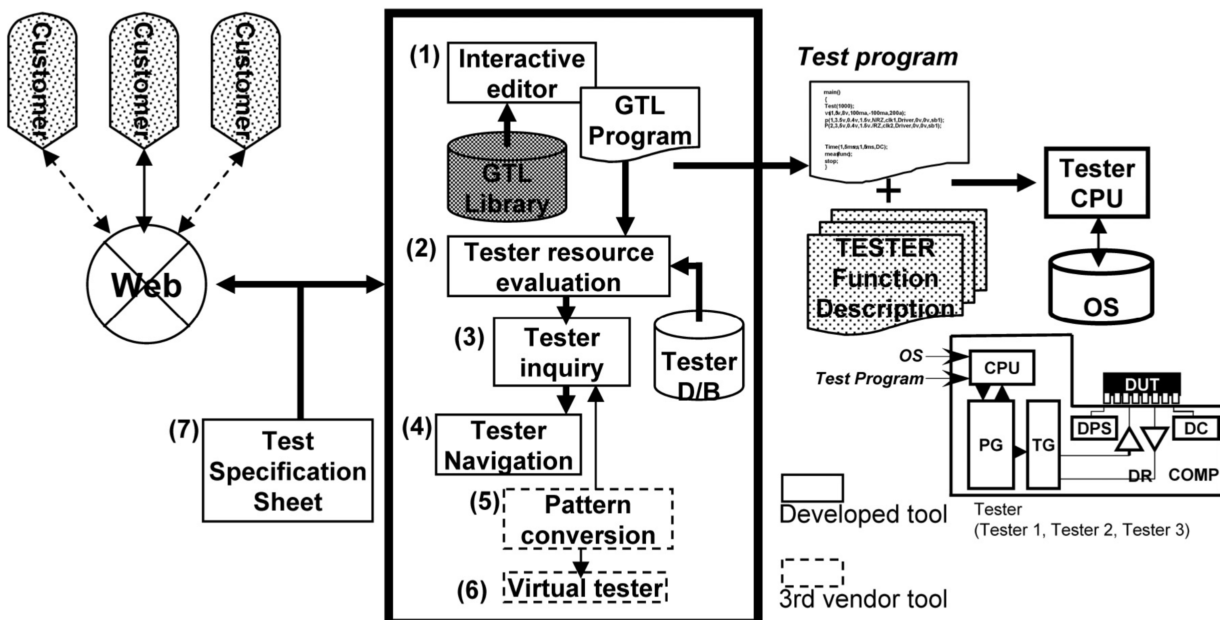

Fig. 8. Prototype environment for the test program development using the GTL.

of the format of the program, such as the order of the parameter setting and the meanings of the parameters.

For example, when a user inputs “VS”, the interactive editor recognizes the name of function, displays the names (meanings) of the six parameters, and prompts the first parameter of the unit number to be input.

The user might not always want to set all of the parameters, due to a lack of knowledge of the parameter or a desire to save time. The interactive editor knows whether each parameter is requisite, and the input of non-requisite parameters can be skipped by inputting a blank (by pressing the space bar). In the example of the VS, only the first and second parameters are necessary, and the remainder of the parameters can be omitted.

(2) Tester resource evaluator

This component extracts the resources required for a GTL program. For example, the following parameters can be derived: number of power supply pins, their voltage range, and number of test-dedicated pins. The extracted data are compared with the specifications of the possible ATE.

(3) Tester inquiry

This module is used to show a list of ATE candidates according to the data extracted from the GTL program. For some candidates, items that do not meet the requirements are displayed. In other words, the tester inquiry can indicate not only the applicable testers, but also inapplicable testers with the resources conflicting with requirements. In addition, the cost of each tester, which is derived from predefined depreciation and administration costs, can be calculated.

(4) Tester navigator

When an inadequate tester is chosen on the tester inquiry, the exact location of the conflicting resources in the GTL test program is revealed. Users can correct the program on the interactive editor, and repeat inquiry. As a result, the range of applicable testers is widened. If the requirements can be relaxed, that is, if the program can be modified, you will be able to get new candidates. Repeatedly modifying the program allows the most cost-effective ATE to be chosen.

(5) Pattern converter

This module converts the test pattern format produced by a logic simulator to value change data (VCD), wave generation language (WGL), or STIL format.

(6) Virtual tester

This software precisely simulates the target VLSI tester [11].

(7) Test specification sheet

A test program can be generated from a specification sheet and so can be used by engineers in fabless companies, who are not familiar with VLSI testers.

4.3 Development Scale

We developed a prototype of the proposed system by Visual C++. Table 3 summarizes the scale of development. The interactive editor was described in 3,700 steps by C++, which is equivalent to 3,700 lines of C++ source code. This took four man-months. Similarly, the tester resource evaluator, tester inquiry, and tester navigator are described in 3,200 steps (two man-months), 2,600 steps (one man-month), and 13,800 steps (five man-months), respectively. We also developed a test specification sheet by Excel VBA. The development of the test specification sheet required 4,500 steps (three man-months). In total, the development of the prototype required 27,800 steps and 15 man-months.

Table 3. Number of man-months and number of lines needed to implement the GTL development environment.

Software	Tool	Man-months	Scale (k steps)
Interactive editor	Visual C++	4	3.7
Tester resource evaluator	Visual C++	2	3.2
Tester inquiry	Visual C++	1	2.6
Tester navigator	Visual C++	5	13.8
Test specification sheet	EXCEL, VBA	3	4.5
Total	-	15	27.8

Using the tester selection tool, the man-hours required to select a suitable ATE can be reduced. For a typical SoC, approximately five engineers gather and discuss the details of the following requirements: pin-count, clock frequency, number of clock phases, power supplies, input/output level, and number of function patterns. By reviewing the ATE manuals, the engineers confirm and reconfirm that the selected ATE meets all requirements. This required two weeks, or approximately twenty hours, in other words 100 man-hours.

Contrary to this conventional method, if the proposed tester selection tool is available for the engineers, it is not necessary to reconfirm the requirements because the tool shows the candidate VLSI testers based on the GTL. It is estimated that approximately two hours are required for one meeting, in other words 10 man-hours. Therefore, the number of man-hours required for selecting a tester can be reduced to 10%.

5. Tester Applicability Evaluation

We evaluated the prototype using real testers. As a DUT, a simple 8-bit adder was implemented on an FPGA chip. The target testers were T6575 and T3324 by Advantest. The test program was to perform, for example, open/short, input-leakage, power consumption, output voltage, and functional tests. The test program was described by 440 lines by the GTL. Appendix B shows the test program and applied test pattern.

Table 4. Execution time for the GTL and an ATE language.

			Test Time	
Test No.	Test Item	T6675 GTL	T3324 GTL	T3324 MOST
-	INIT	-	86.30ms	87.10ms
100	OPEN	-	171.8ms	172.4ms
200	SHORT	-	517.9ms	515.3ms
300	LEAK-H	-	711.0ms	705.7ms
400	LEAK-L	-	901.2ms	892.8ms
700	STRESS	-	958.6ms	937.0ms
800	VIH/VIL MIN	-	1.006s	972.8ms
900	VOH	-	1.135s	1.096s
1000	VOL	-	1.264s	1.217s
2000	ICC	-	1.326s	1.262s
2100	Istby	-	1.382ms	1.319s
3000	VCCMIN	-	1.430s	1.355s
3100	VCCMAX	-	1.475s	1.391s
3200	VCCTYP	-	1.519s	1.427s
4000	ACC(VCCMIN)	-	1.579s	1.472s
4100	AC(VCCMAX)	-	1.633s	1.517s
4200	AC(MAX-MIN)	-	1.641s	1.524s
5000	FERQ(VCCMIN)	-	1.705s	1.577s
5100	FERQ(VCCMAX)	-	1.769s	1.631s
5200	FERQ(MAX-MIN)	-	1.777s	1.639s
6000	PERIOD(VCCMIN)	-	1.841s	1.692s
6100	PERIOD(VCCMAX)	-	1.905s	1.745s
6200	PERIOD(MAX-MIN)	-	1.913s	1.753s
Total Test Time		1.719s	1.921s	1.761s

Figure 9 shows the environment and a shmoo plot with T6575. Since the tester and DUT were connected using an inexpensive cable due to a very limited test budget, some noises appeared because of the unmatched impedance.

We compared the test time of the GTL program with that of the program written in the original language of the testers. Table IV shows the results of the test time with T6575+GTL, T3324+GTL, and T3324+MOST, where MOST is the tester language that is proprietary to Advantest Corp. Twenty-three test items are compared in the table. Generally speaking, in comparison to the test program written in the original language of the testers, MOST, the execution time with the GTL is approximately the same as that of MOST. One drawback of the GTL is the longer compilation time due to the use of a higher-level language. As a trade-off, the GTL improves readability and portability.

6. Conclusions

In the present paper, we proposed a tester structure expression language called the general tester language (GTL), to improve the readability and portability of tester programs based on the observation that tester languages are incompatible with each other, while the structure of VLSI testers is approximately the same. The tester resources are implemented as a set of functions written in the C++ language. We also developed a prototype of the test solution environment based on the proposed language. Its program size is approximately 27,800 steps and 15 man-months were required. The number of man-hours for the tester selection process could be reduced by 1/10 for typical SoCs. Experimental results show the test time for a program written by GTL is approximately the same as that written by the proprietary language. In the future, we intend to construct a test service that is available via the Internet.

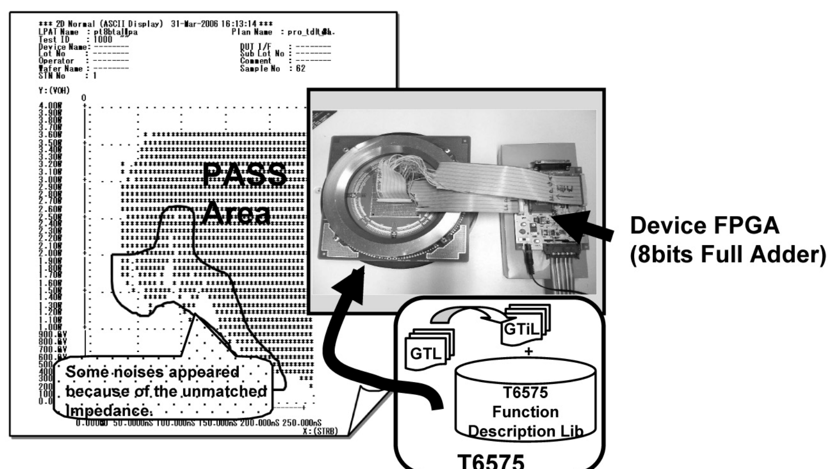

Fig. 9. Evaluation setup and results obtained using the T6575 (Advantest).

Acknowledgements

The authors would like to thank the members of Working Group 2 (Test engineering) of Semiconductor Technology Roadmap of Japan (STRJ) for their contribution to the tester language survey. The authors also would like to thank Prof. Satoshi Fukumoto of Tokyo Metropolitan University for his helpful comments.

References

[1] Semiconductor Industry Association, International Technology Roadmap for Semiconductors. 2007. http://www.itrs.net/

[2] T. Kazamaki, H. Maruyama, and S. Sumida, “Trial Model of 100 MHz Test Station for High Speed LSI Test System,” International Test Conference, pp. 611-617, 1978.

[3] T. Kazamaki, “A 100MHz Tester - Challenge to New Horizon of Testing High Speed LSI,” International Test Conference, pp. 618-625, 1979.

[4] S. Bisset, “The Development of a Tester-per-Pin VLSI Test System Architecture,” International Test Conference, pp. 151-157, 1983.

[5] J. Katz and R. Rajsuman, “A New Paradigm in Test for the Next Millennium,” International Test Conference, pp. 468-476, 2000.

[6] J. Bedsole, R. Raina, A. Crouch, and M. S. Abadir, “Very Low Cost Testers: Opportunities and Challenges,” IEEE Design & Test of Computers, Vol.18, No.5, pp. 60-69, Sep.-Oct. 2001.

[7] K. Posse and G. Eide, “Key Impediments to DFT-Focused Test and How to Overcome Them,” International Test Conference, pp. 503-511, 2003.

[8] M. Sato, N. Otsuka, O. Muto, M. Arai, S. Fukumoto, K. Iwasaki, K. Uehara, I. Shimizu, and H. Mamyouda, “Development of Low-Power Board-Mounted Reconfigurable Tester,” IEICE Transactions on Information and Systems, Vol. J88-D-I, No. 6, pp. 1065-1075, June 2005. (in Japanese)

[9] J386A/J386A-8 Customer Service Manual, Teradyne, April 1985.

[10] 8311122, VLSI Test System (ATL-30) Programming Handbook, Advantest, 1998.

[11] M. Sato, “Memory Virtual Tester Technology and a Tester on Chip,” SEMI Technology Symposium, pp. 70-75, December 2000.

[12] IEEE Standard Test Interface Language (STIL) for Digital Test Vector, IEEE Std. 1450-1999.

[13] Japan Electronics and Information Technology Industries Association, Semiconductor Technology Roadmap Committee of Japan, 2003.

[14] M. Sato, H. Wakamatsu, and K. Iwasaki, “Investigation of a Tester Language and a Tester Structure Expression Language,” SEMI Technology Symposium, pp. 4.21-4.24, December 2000.

[15] Test Systems Strategies, Inc., http://www.tssi.com/

Masayuki Sato
He worked for Hitachi Ltd. as a test engineer for VLSI chips from 1971 to 2002. After that he worked for Innotech Corp. from 2002 to 2004 and worked for Genesis Technology Inc. from 2004 to 2007. From 2008 he has worked for Taiyo Yuden Co. Ltd. and is presently a manager of Intellectual Property Department. From 2003 he has been a Ph.D. Student of Tokyo Metropolitan University. He is a member of the IEICE.

Hiroki Wakamatsu
He worked for Sanwa Koki Co. Ltd. from 1988 to 2005 and worked for Genesis Technology Inc. from 2005 to 2007. From 2008 he has worked for Oki Engineering Co. Ltd. as a test engineer.

Masayuki Arai
He received his B.E., M.E, and Ph. D. degrees from Tokyo Metropolitan University in 1999, 2001, and 2005, respectively. Currently he is an assistant professor of Tokyo Metropolitan University. His research areas include dependable computing and VLSI testing. He is a member of the IEEE and IEICE.

Kenichi Ichino
He received the M.E. and Ph.D. degrees from Tokyo Metropolitan University in 2001 and 2004, respectively. He is currently a Visiting researcher of Tokyo Metropolitan University. His research interest includes built-in self-test and design for testability.

Kazuhiko Iwasaki
He received the B.E. degree in 1977, the M.E. degree in 1979, and the Ph.D. degree in 1988, all from Osaka University. He joined Hitachi's Central Research Laboratory in 1979, where he developed VLSI processors. From 1990 to 1995 he was an associate professor of Chiba University. Presently, he is a professor of Tokyo Metropolitan University. His research interests include dependable networking and VLSI testing. He is a senior member of the IEEE and a member of the IEICE.

Takeshi Asakawa
He received the B.E. degree from Tokai University in 1984, and the Ph.D. degree from Tokyo Metropolitan University in 2000. He joined NEC IC micro computer systems in 1984, where he engaged in the development of video memories. He is currently an associate professor of Tokai University. His research areas include design for testability and computer architecture. He is a member of the IEEE.

Appendix A

(Tester 4)
uvi(VCC) {v = @vcc; vi = vfim; i = 3 mA; vrng = r8V; irng = r3mA; alarm = off; }

This ATE was developed in 1998 and has a C-like language. The name of the tester resource is denoted by VCC, and values in {} are parameters for the resource. The first term, v = @vcc, is the applied voltage. The second term, vi = vfim, means that the voltage is applied and the current is measured. The third term, i = 3 mA, is the clamp current to protect the circuit. The forth term, vrng = 8 V, denotes the voltage range. The fifth term, irng = r3 mA, is the current range. The last term, alarm = off, prevents the alarm signal from occurring, even if the current overflows the clamp value.

(Tester 5)
set vs1 v = 0.000v, vr = 10v, cur = 400ma, ir = 800ma;

This tester was developed in 2000 and has a C-like language. The resource is designated as vs1. The following are the parameters of the resource. The first term, v = 0.000v, is the applied voltage. The second term, vr = 10v, is the voltage range. The third term, cur = 400ma, is the clamp current. The last term, ir = 800ma, is the current measurement range.

Appendix B: Example of a GTL test program

(a) Test program for 8-bit adder (extract)

```
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/_/_/          MAIN PROGRAM          _/_/_/_/_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
MAIN(8bit_adder)

//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/          UNIT          /_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
TIME(1,3mS,"VS");
TIME(2,6mS,"ALL");

//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/      Contact Test      /_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
TEST(100);
VS(1,0V,8V,2A,1A,-1A);
ISVM(-100uA,800uA,8V,3V,-3V);
LIMIT("DC",100mV,-1.5V);
SRON();
MEAS("DC",ALLPIN);
SROFF();

//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/          RATE          /_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
RATE(1,50nS*5);
```

```
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/              PINS              /_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
PIN(ADD_A,"IN" ,"NRZ1", 1, 0nS, 0nS, 0nS, 0nS, "DRENRZ", 0nS, 0nS, 5V, 0V, 0V, 0V         0nS, 0nS);
PIN(ADD_B,"IN" ,"NRZ1", 1, 0nS, 0nS, 0nS, 0nS, "DRENRZ", 0nS, 0nS, 5V, 0V, 0V, 0V,        0nS, 0nS);
PIN(OUT_Y,"OUT","NON",    1, 0nS, 0nS, 0nS, 0nS, "DRENRZ", 0nS, 0nS, 0V, 0V, 3V, 1V, 47nS*5, 0nS);
PIN(CARY, "OUT","NON",    1, 0nS, 0nS, 0nS, 0nS, "DRENRZ", 0nS, 0nS, 0V, 0V, 3V, 1V, 47nS*5, 0nS);

//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/       Function Test        /_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
TEST(1000);
SRON();
MEAS("FT","pt8bti.lpa");
MEAS("FT","pt8bta.lpa");
MEAS("FT","pt8btall.lpa");
SROFF();

//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
//_/_/              FINISH            /_/_/
//_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/_/
STOP();
ENDMAIN()
// program end
```

(b) Test pattern (extract)

```
LPAT PT8BTA
RDX 10
CHANNEL     26, 8-1,         16-9,        24-17,     25
CFPF
LOC 0
;               C   AAAAAAAA   BBBBBBBB   YYYYYYYY   C
;               L   76543210   76543210   76543210   A
;               K                                    R
;                                                    Y
;
NOP /T1 !   1   00000000   00000000   XXXXXXXX   X ;STEP0   INIT
NOP /T1 !   1   00000000   00000000   LLLLLLLL   L ;STEP1   0+0=0
NOP /T1 !   1   00000000   00000001   LLLLLLLH   L ;STEP2   0+1=1
NOP /T1 !   1   01111110   01111110   HHHHHHLL   L ;STEP3   126+126=252
NOP /T1 !   1   01111111   01111111   HHHHHHHL   L ;STEP4   127+127=254
NOP /T1 !   1   10000000   10000000   LLLLLLLL   H ;STEP5   128+128=256 (0)
NOP /T1 !   1   10000001   10000001   LLLLLLHL   H ;STEP6   129+129=258 (2)
NOP /T1 !   1   11111110   11111110   HHHHHHLL   H ;STEP7   254+254=508 (252)
NOP /T1 !   1   11111111   11111111   HHHHHHHL   H ;STEP8   255+255=510 (254)
NOP /T1 !   1   00000000   00000000   XXXXXXXX   X ;STEP9   INIT

STPS
END
```