Development of Extended-STIL Pattern Compiler for Test Programming Environment

Seong-Jin Kim
Dept. of Computer Science and Engineering
Sangji University
Wonju-si Gangwon-do, Republic of Korea
Seongjin.gim@gmail.com

Kwang-Man Ko
Dept. of Computer Science and Engineering
Sangji University
Wonju-si Gangwon-do, Republic of Korea
kkman@sangji.ac.kr

***Abstract*—In this paper, we extended the blocks that can generate analog signals in the Alternating Current (AC) Instrument board along with the basic blocks provided by the Standard Test Interface Language (STIL) standard. We developed a compiler that extracts test information by analyzing STIL files by block. In order to verify the accuracy of the test information extracted by the compiler, experimental results using Automatic Test Equipment (ATE) are presented.**

Keywords—STIL pattern compiler; Standard test interface language; test pattern

I. INTRODUCTION

Standard Test Interface Language (STIL) is a standard test interface language that allows one to define test patterns and waveforms, which are generated when simulating digital integrated circuits, in one language. STIL provides an interface between digital test generation tools and the test equipment [1]. It can be created directly as a test generation tool's output language or used as an intermediate format for processing a specific stage. Therefore, STIL is ideal for exchanging data between a Computer-Aided Design (CAD) or a simulator and a test environment. Test programs written in STIL generally consist of seven blocks that define STIL, Signals, SignalGroups, Timing, PatternBurst, PatternExec, and Pattern. Each block is created based on the design description of the chip to be tested [2].

In this paper, we expanded the functions of STIL by adding blocks that can be applied to AC test instruments board along with the basic blocks provided by the STIL standard and have developed a compiler that extracts and saves the necessary test information by interpreting the file created with the extended STIL. In addition, we utilized the digital instrument board for open/short test which is a DC parameter test. We also experimented for gross function test to verify the truth table for Texas Instruments' SN74LS00N chip in order to validate the test information extracted by the compiler.

II. BACKGROUNDS AND MOTIVATIONS

A. Standard Interface Language and Block

STIL is a standard language that provides an interface between digital test generation tools and test equipment. It is either created directly in the output language of the test generator or is used as an intermediate format for specific stage processing. In addition, STIL is well suited as data exchanged between a CAD or simulator and a test environment [3][4].

STIL files generated by an ATPG or simulator are used as inputs to a converter or a compiler to classify and store test information for each component. Among the stored information, the test vector is loaded into the memory of the target tester when necessary. The test vector is the most important element defined in the STIL language. It is used to detect defects and is usually similar to the truth table type and consists of input data and output data. The test vectors and patterns are used in combination and are used to measure the logic functions and AC/DC functions of semiconductor products. The STIL file is also used as an input to the STIL manipulation tool and can be used as an output to generate STIL files with specific rules and commands added. STIL allows tester-dependent programs to be applied to specific ATE systems and directly connects ATPG tools such as CAD/CAE to the ATE environment. A test program written in this STIL is generally composed of 7 blocks, and each block is created based on the design specification of the chip to be tested.

B. Motivations and Contributions

Developers of ATE (Automatic Test Equipment) that can test and evaluate digital devices and provide an easy-to-access debug-able test program environment for users to test various devices. Users who write test programs are required to become familiar with the structure and behavior of test languages and ATE, and it is especially important to write and maintain test patterns used to evaluate and test DUT (Device Under Test) during testing. Since the use of a test description language is essential to easily create and manage these test patterns, many ATE vendors are using STIL standards that can easily describe the structure and test patterns of DUTs. Therefore, it is essential that the STIL standard used in the test description language is software that classifies and stores test information for each component by analyzing STIL files so that it can be interpreted by the test equipment [5].

Typically, the ADVANTEST SoC test systems T2000 and V93000 generate STIL files of patterns, timing, and level information generated by ATPG as shown in the Figure 3, and provide STIL Reader to convert the generated STIL files to the test system. TERADYNE's mixed-signal SoC test systems, UltraFLEX and J750, also provide IG-XL Test Software, which converts STIL, WGL (Waveform Generation Language) and VCD (Value Change Dump). The use of the STIL standard is essential for testing a variety of devices using ATE, and a compiler for interpreting and categorizing files written in the STIL standard is a must-have tool [6].

III. EXTENDED STIL

A. *Abbreviations and Acronyms*

The SourceWave block is a block that allows the user to create waveforms by defining eight kinds of information, such as period, frequency, and amplitude for four types of waveforms such as Sine, Ramp, Pulse, and Staircase. To do this, we define keywords and token groups that can be defined in the SourceWave block. A total of 15 keywords can be used in the SourceWave block and four types of waveforms can be defined by the user. The waveforms are divided into five token groups according to the attributes of the keywords. The division of a keyword into token groups is a way to make it as simple and easy to use as possible when defining a grammar,

B. *SourceWave Definition*

Fig. 1 shows the BNF representation of the SourceWave block syntax and how it can actually be defined based on these BNF representations.

```
source_wave ::= SourceWave source_wave_name "{"
                  [outer_wave_item_list]
                  signal_wave
                "}"
source_wave_name ::= identifier
outer_wave_item_list ::= outer_wave_item
                   | outer_wave_item_list outer_wave_item
signal_wave ::= signal_name "{"
                  wave_type "{"
                    [inner_wave_item_list]
                 "}"
                 [Operation type_operand]
                "}"
inner_wave_item_list ::= inner_wave_item
                  | inner_wave_item_list outer_wave_item
outer_wave_item ::= outer_current_type type_operand ";"
inner_wave_item ::= `inner_current_type type_operand ";"
outer_current_type ::= Period | Frequency | Sample
inner_current_type ::= Period | Amplitude | Offset
                     | Repeat | Phase | Invert
                     | MaximumRise
type_operand ::= integer | float
signal_name  ::= identifier
wave_type    ::= Sin | Ramp | Pulse | Staircase
```

Fig. 1. BNF Representation of SourceWave Block

In Fig. 1, the SourceWave block can define a signal wave with one or more outer wave items and only one signal name, either the signal or the signal name defined in the SignalGroups block. In the "outer wave" item, we can declare a type and type subject with three keywords: period, frequency, and sample. In the waveform definition of the signal, we can declare 8 attributes and 4 waveforms in total.

IV. STIL PATTERN COMPILER

The STIL Pattern Compiler implemented in this study receives a STIL file composed of basic blocks provided by the STIL standard as input. The STIL Pattern Compiler analyzes the file and stores the necessary test information. In addition, the functions were extended by defining additional blocks applicable to AC/DC instruments, and an intermediate file that can be recognized by the test program and the instrument is generated through the linker and the loader.

A. *Overall Structure*

The overall structure of the STIL Pattern Compiler is shown in Fig. 2. The STIL Pattern Compiler is divided into Small Vector Compiler, Small Vector Linker, and Small Vector Loader. Small Vector Compiler and Linker operate in conjunction with Test Program which defines a STIL file, and Loader works in conjunction with Test OS to send test data to the instrument.

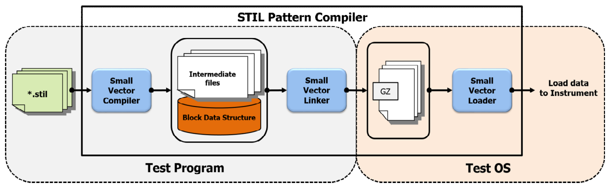


Fig. 2. The Structure of STIL Pattern Compiler

First, the Small Vector Compiler receives a STIL file as an input, classifies and analyzes the data by blocks, stores the analyzed data in a block data structure, and generates an intermediate file as the input of the linker. Second, the Small Vector Linker receives the intermediate file generated by the Small Vector Compiler as input and generates an input file of the Small Vector Loader. The intermediate files received as inputs at this time contain pattern and block information. Finally, the Small Vector Loader receives a compressed file generated by the Small Vector Linker as input and loads the test data in the actual instrument.

B. *Small Vector Compiler*

The main purpose of the STIL Pattern Compiler is to analyze and classify incoming STIL files and provide the required test data to the Test OS and Instrument. The structure of Small Vector Compiler designed and implemented in this study to perform this function is shown in Fig. 3.

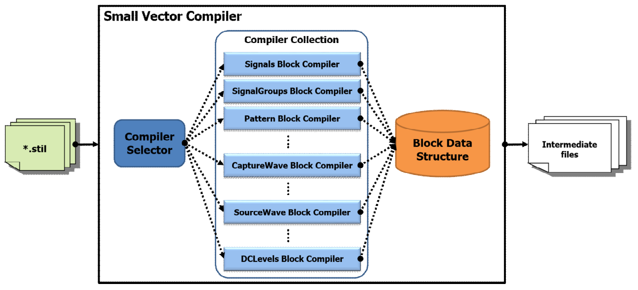


Fig. 3. The Structure of Small Vector Compiler

Small Vector Compiler is divided into Compiler Selector that receives STIL files as inputs and calls corresponding Compiler, Compiler Collection area that analyzes and stores STIL blocks, and Block Data Structure that stores test data. First, the Small Vector Compiler delivers the user-defined STIL files received as inputs to the Compiler Selector. Compiler Selector divides the received STIL files into blocks and calls the corresponding Block Compiler in the Compiler Collection area consisting of 18 Block Compilers to analyze the Block contents. The called Block Compiler extracts the test data while analyzing the block contents, stores it in the block data structure, and generates an intermediate file for storing the pattern in-formation and the test data. The Block Data Structure stored by the Small Vector Compiler contains a lot of test data which is used as very important data in the Test OS. The intermediate file is used as the input of the Small Vector Linker. The core of the Small Vector Compiler implemented in this study is the Compiler Collection area, which is a set of block compilers that analyze each block and store test data.

C. Block Compiler

Block Compiler in the Compiler Collection area basically analyzes the corresponding STIL file and stores necessary data, and has the structure shown in Fig. 4 to perform such functions.

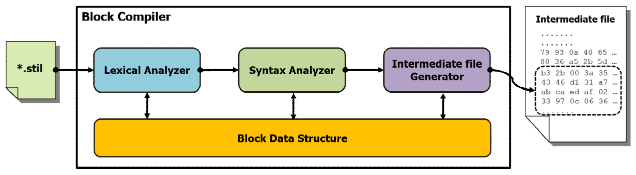


Fig. 4. The Structure of Block Compiler

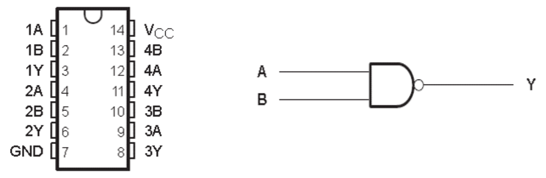
Block Compiler, which receives a STIL file as an input, first divides it into tokens, which are grammatically meaningful minimum units, through a lexical analyzer. In order to do this, we have defined regular expressions and state transitions, and implemented a recognizer to identify all the tokens that are needed in the STIL file that was received as input. In the next step, the parser receives the tokens identified in the lexical analysis step, checks errors against the syntax defined in the STIL file, and extracts the test data information if there is no error. At this time, the extracted data is stored in the Block Data Structure and is input to the intermediate file generator. In the final step, the inter-mediate file generator generates an intermediate file, which is the input of the Small Vector Linker, in a specific format based on the test data information analyzed by the parser.

V. EXPERIMENTS AND RESULTS

In order to verify the operation of the STIL Pattern Compiler implemented in this study and the accuracy of extracted test data, open/short test, which is a DC parameter test, and gross functional test were performed for verification of defects in the circuit using the Digital Instrument Board [7], which is used in ATE [8], with SN74LS00N, which is a 4 channel NAND gate IC of Texas Instruments. For this purpose, STIL and a test program for chip operation were prepared, compiled, and loaded into TestOS and the instrument board, and the results of the two tests were obtained. In addition, the information extracted from STIL was verified using the STIL Viewer Tool of TestOS.

A. Test Program Configuration

In order to drive and test the chip, it is necessary to write and compile the test program. For this, the STIL must be defined first based on the chip specification. Fig. 5 shows the pin configuration, logic diagram, and truth table among the chip specifications used for the experiment in this study.



(a) Package (b) Logic diagram

FUNCTION TABLE
(each gate)

INPUTS		OUTPUT
A	B	Y
H	H	L
L	X	H
X	L	H

(c) Function table

Fig. 5. The Specification of SN74LS00N

In this figure, the SN74LS00N consists of 14 pins (8 inputs, 4 outputs, Vcc, and GND) and 4 NAND gates. The output has a low value only when the inputs are all high.

```
STIL 1.0;
Signals {
  "ina0"  In; "ina1" In; "outa" Out;
  "inb0"  In; "inb1" In; "outb" Out;
  "outc" Out; "inc0" In; "inc1"  In;
  "outd" Out; "ind0" In; "ind1"  In;
  "vcc"   In;
}
SignalGroups {
  ainput = 'ina0 + ina1';
  binput = 'inb0 + inb1';
  cinput = 'inc0 + inc1';
  dinput = 'ind0 + ind1';
  input  = 'ainput + binput + cinput + dinput';
  output = 'outa + outb + outc + outd';
  all = 'input + output';
}
Timing "timeNormal" {
  WaveformTable "ts0" {
    Period '100ns';
      Waveforms {
        all {
          P { '1.0ns' P; }
          0 { '0.0ns' N; '1.0ns' D; }
          1 { '0.0ns' N; '1.0ns' U; }
          L { '0.0ns' Z; '81.0ns' L; }
          H { '0.0ns' Z; '81.0ns' H; }
          X { '0.0ns' Z; '81.0ns' X; }
          T { '0.0ns' Z; '81.0ns' T; }
          Z { '0.0ns' Z; }
          2 { '0.0ns' 2; }
        } // end all
      } // end waveforms
  }  // end WaveformTable
} // end Timing
PatternBurst "burst" {
  PatList {
    "gross";
  }
}
PatternExec "exec" {
  Timing       "timeNormal";
  PatternBurst "burst";
}
Vector gross {
  Signals ("all");
  W ts0;
  > XX XX XX XX X X X X;
  loop_start 1000
  > 00 01 10 11 H H H L;
  > 01 10 11 00 H H L H;
  > 10 11 00 01 H L H H;
  > 11 00 01 10 L H H H;
  > 10 11 00 01 H L H H;
  > 01 10 11 00 H H L H;
  loop_end;
  stop;
}
```

Fig. 6. The Definition of STIL for SN74LS00N

Base on this, the STIL for chip test was defined as shown in Fig. 6. First, the signal name and type for the input and output pins of 4 NAND gates were declared in the signal block and the signals defined above were classified into seven groups in the SignalGroups block for convenience. In the Timing block, a WaveformTable with the name ts0 was defined with nine WaveformChars for a period of "100ns" and "all" groups. In addition, timing and pattern names to be used were described in the PatternBusrt and PatterExec blocks. Finally, in the Vector (block) block, the operation of actual signals was defined using the signal and timing block information.

B. *Experimental Environment*

After compiling the STIL file and test program created in Section 4.1 and loading the test data to TestOS and the Digital Instrument Board, the experimental environment as shown in Figure 16 was constructed to test the characteristics and defects of the corresponding device using the device verification tool.

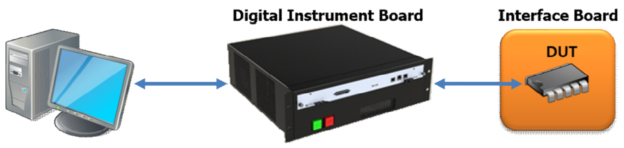


Fig. 7. The Experimental Environment

First, TestOS is connected to the Digital Instrument Board to load the compiled test program and test data. The Digital Instrument Board used in the experiment has 64 I/O channels with basic 200Mhz/400Mbps speeds and provides 32 timing sets and 4 edges per channel. Next, an interface board was used for connection between the Digital Instrument Board and the DUT as shown Fig. 8. A total of five devices can be connected to this interface board and two SN74LS00N chips were connected in this study.

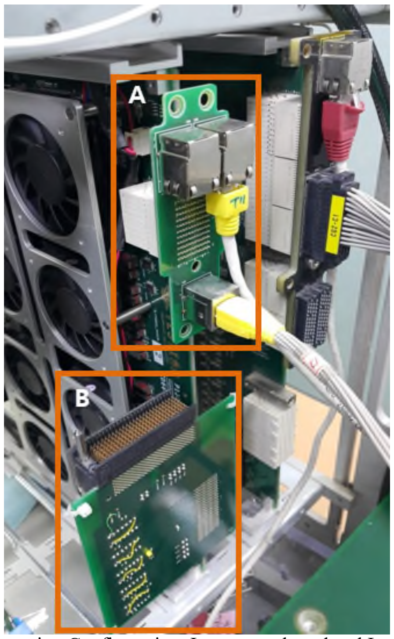
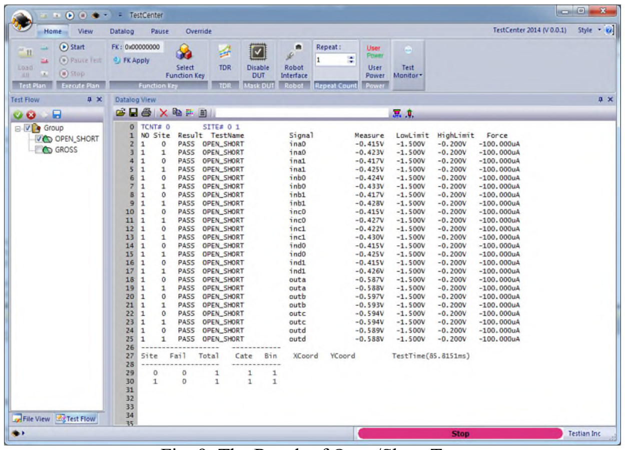


Fig. 8. The Connection Configuration: Instrument board and Interface board

C. *Experiment Result*

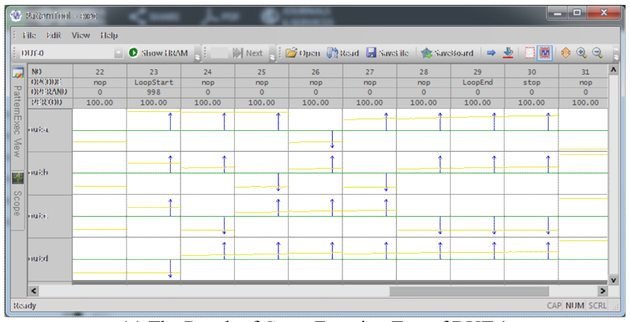
In the first experiment, open/short test, which is a DC parameter test, was per-formed to verify that the DUT operates normally in the specified environment. TestCenter Tool, an engineering tool for device verification, was used to execute the test defined in the test program and analyze the characteristics of the device. The result of the experiment is shown in Figure 18. From the result, we can see that the measured values of all the signals defined in the STIL file are between -0.2 and -1.5 V, con-firming that all the pins of the two tested devices are normal.



```
TCNT# 0         SITE# 0 1
NO Site  Result  TestName          Signal        Measure   LowLimit  HighLimit  Force
1    0   PASS   OPEN_SHORT         ina0          -0.415V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         ina0          -0.423V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         ina1          -0.417V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         ina1          -0.425V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         inb0          -0.424V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         inb0          -0.433V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         inb1          -0.417V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         inb1          -0.428V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         inc0          -0.415V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         inc0          -0.427V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         inc1          -0.422V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         inc1          -0.430V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         ind0          -0.415V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         ind0          -0.425V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         ind1          -0.415V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         ind1          -0.426V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         outa          -0.587V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         outa          -0.588V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         outb          -0.597V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         outb          -0.593V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         outc          -0.594V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         outc          -0.594V   -1.500V   -0.200V   -100.000uA
1    0   PASS   OPEN_SHORT         outd          -0.589V   -1.500V   -0.200V   -100.000uA
1    1   PASS   OPEN_SHORT         outd          -0.588V   -1.500V   -0.200V   -100.000uA
-------------------   ------------
Site   Fail   Total    Cate   Bin    XCoord   YCoord        TestTime(85.8151ms)
-------------------   ------------
  0      0      1        1     1
  1      0      1        1     1
```

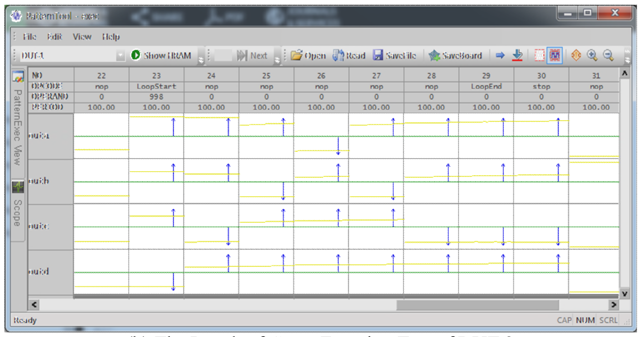
Fig. 9. The Result of Open/Short Test

In the second experiment, a gross functional test was performed to verify the presence of defects on the DUT circuit. To obtain the execution result of the test, we used the Pattern Tool which shows the pattern information defined in the PatternExec block and the actual execution result of the pat-tern and the waveform of each signal. The result of the experiment is shown in Fig. 10. The yellow line in the result signifies the response data output from the DUT, and the blue line signifies the expected data to be compared with the DUT output. The green line is the baseline of the data, and the upper side of the line signifies HIGH and the lower side LOW. Figure 10(a) and 10(b) show the wave-forms of the response data output by each DUT and the expected data for output signals, verifying that the waveform of the response data from each DUT matches that of the expected data. There-fore, the two tested devices have no problem in the circuit.



NO	22	23	24	25	26	27	28	29	30	31
OPCODE	nop	LoopStart	nop	nop	nop	nop	nop	LoopEnd	stop	nop
OPERAND	0	998	0	0	0	0	0	0	0	0
PERIOD	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00

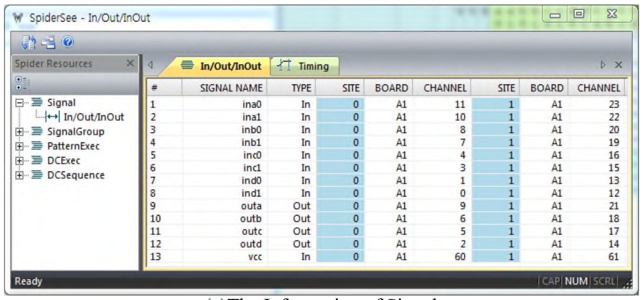
(a) The Result of Gross Function Test of DUT 1



(b) The Result of Gross Function Test of DUT 2

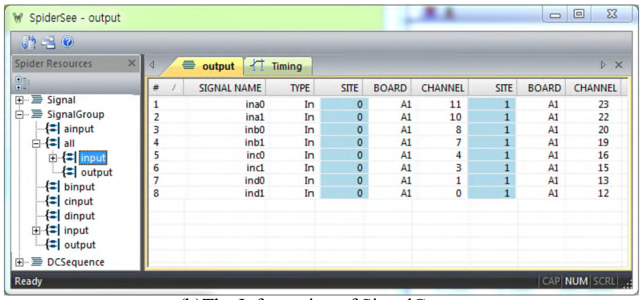
Fig. 10. The Result of Gross Function Test

Finally, the information extracted by the STIL Pattern Compiler was confirmed by using the STIL Viewer Tool of TestOS, as shown in Fig. 11. The Viewer Tool shows block information such as Signal (Fig. 11(a)), SiganlGroup (Fig. 20(b)), and Timing (Fig. 20(c)) information defined in the STIL, and it can be seen that the test information extracted by the compiler is normally stored.



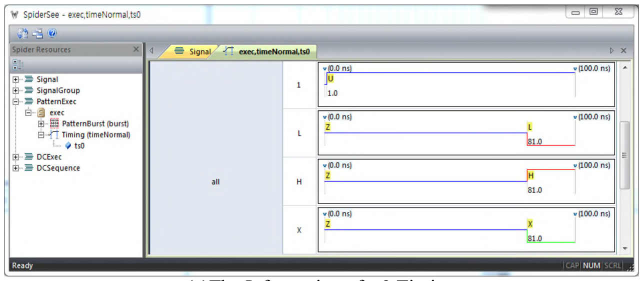
#	SIGNAL NAME	TYPE	SITE	BOARD	CHANNEL	SITE	BOARD	CHANNEL
1	ina0	In	0	A1	11	1	A1	23
2	ina1	In	0	A1	10	1	A1	22
3	inb0	In	0	A1	8	1	A1	20
4	inb1	In	0	A1	7	1	A1	19
5	inc0	In	0	A1	4	1	A1	16
6	inc1	In	0	A1	3	1	A1	15
7	ind0	In	0	A1	1	1	A1	13
8	ind1	In	0	A1	0	1	A1	12
9	outa	Out	0	A1	9	1	A1	21
10	outb	Out	0	A1	6	1	A1	18
11	outc	Out	0	A1	5	1	A1	17
12	outd	Out	0	A1	2	1	A1	14
13	vcc	In	0	A1	60	1	A1	61

(a)The Information of Signal



#	SIGNAL NAME	TYPE	SITE	BOARD	CHANNEL	SITE	BOARD	CHANNEL
1	ina0	In	0	A1	11	1	A1	23
2	ina1	In	0	A1	10	1	A1	22
3	inb0	In	0	A1	8	1	A1	20
4	inb1	In	0	A1	7	1	A1	19
5	inc0	In	0	A1	4	1	A1	16
6	inc1	In	0	A1	3	1	A1	15
7	ind0	In	0	A1	1	1	A1	13
8	ind1	In	0	A1	0	1	A1	12

(b)The Information of SignalGroup



(c)The Information of ts0 Timing

Fig. 11. The Information of STIL Viewer Tool

VI. Related Works

STIL-based pattern generation tools eliminate the process of switching to a specific format of the tester by directly connecting ATPG tools with ATE [9]. Compatible with the ADVANTEST T2000 system, OPENSTAR ™ has established an efficient communication link between the Electronic Design Automation (EDA) system and the ATE platform. In addition, they applied the same standards as STIL, Core Test Language (CTL), and Standard for Embedded Core Test (SECT) to support common solutions [10]. Teradyne's Ultra FLEX digital instruments were applied to the Test Insight Tool Suite [11]. Test Insight is the primary partner for Teradyne and Advantest, offering a variety of tools for testing and validating test conversions, testing programs [12]. In particular, ATEGEN, a test program generator, generates tester program files for various ATE formats such as general IC tester, J750, and 93K for files written in WGL and STIL. STILVerify, the STIL Checker for Mentor Graphics Tessent®, is commercially available as a tool for parsing and verifying files written in the STIL standard. STILVerify provides Verilog test bench functionality by checking that files written in STIL are syntactically correct, and by running them in the Verilog simulator to verify the contents and behavior of the code [13]. Synopsys' TetraMax ™ ATPG provides test patterns in STIL format and is a tool that automatically generates high-quality test patterns to reduce mistakes in test patterns created to test complex logic [14]. The STIL Director of Toshiba Microelectronics Corporation is a system tool for building test environments based on the STIL standard and is available as a Toshiba STIL design kit. Because it is an open system, it can be easily applied to specific system environment by plug-in method and can be customized in various environments by using Access interface [15].

VII. Conclusion

In this study, we defined the blocks provided by the STIL standard and extended the functions of STIL by adding blocks applicable to the AC/DC test instruments. We also developed the STIL Pattern Compiler, which can extract and save the necessary test information, by analyzing the file created by STIL. Open/short test and gross functional test were performed on the SN74LS00N chip of Texas Instruments to verify the operation of the implemented compiler and the extracted test information. For the verification experiment, first, the STIL file and the test program were created. Second, the experimental environment was constructed using the Interface Board which connects the Digital Instrument Board, DUT and Digital Instrument Board used in TestOS and ATE. Finally, open/short test and gross functional test were conducted by loading a test program from TestCenter, a device verification tool. The logs provided by the TestCenter and the results of the Pattern Tool confirmed that all pins of the two tested devices were normal and that there were no defects in the circuit.

Acknowledgement

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning (20173030223).

References

[1] IEEE Computer Society, IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data Language Manual IEEE Std. 1450.1999, IEEE New York (1999)

[2] D. Fan et al, Case Study-Using STIL as Test Pattern Language, NPTest, Inc. LLC, ITC International Test Conference, pp. 811-817, 2003.

[3] P. Wohl and N. Biggs, P1450.1: STIL for the simulation environment, VLSI Test Symposium, 18th IEEE, 2000.

[4] A. Pramanick, R. Krishnaswamy, M. Elston, T. Adachi, Harsanjeet Singh, B. Parnas, and L. Chen, Test programming environment in a modular, open architecture test system, ITC 2004 International, 2004

[5] M. Sato, H. Wakamatsu, M. Arai, K. Ichino, K. Iwasaki, and T. Asaka, Tester Structure Expression Language and its Application to the Environment for VLSI Tester Program Development, Journal of Information Processing Systems pp.121-132, 2008.

[6] ADVANTEST, STILReader, [online] https://ebiz.advantest.com/aac/EProductSheets/viewDatasheets.html?id=36&menu=soctab

[7] TESTIAN, TA15 (AC Test Instrument Board), [online]
http://www.testian.co.kr/eng/_products_05.htm

[8] TESTIAN, Spider Nano (Desktop ATE), [online] http://www.testian.co.kr/eng/_products_02_1.htm

[9] H. Lang, B. Pande, and H. Ahrens, Automating test program generation in STIL-expectations and experiences using IEEE 1450 [standard test interface language], The Eight IEEE European, pp. 99–104, 2003.

[10] Y. Ma, "Open Architecture Software for OPENSTAR™ Test Platform," IEEE Future of ATE (FATE) Workshop, Charlotte, N.C., 2003.

[11] TERADYNE Teradyne Software Solutions, [online] http://www.teradyne.com/services/software

[12] Test Insight, ATEGen, http://www.testinsight.com/products/design-to-tester-conversion/test-program-generator.aspx

[13] Mentor Graphics, STIL Checker, [online] https://www.mentor.com/products/silicon-yield/request?&fmpath=/products/silicon-yield/stil_checker&id=73528dbf-95b3-41cc-b9d6-a114a4286ecc

[14] Synopsys, TetraMAX ATPG, [online] https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/test-automation/tetramax-atpg.html

[15] Toshiba Microelectronics, STILDirector, http://www.tosmec-web.toshiba.co.jp/stildirector/eng/products/stildirector1.html