A novel approach to Reusable Time-economized STIL based pattern development

Rahul Malhotra*, Sujay Deb†, Fabio Carlucci‡
*IIIT Delhi, Email: rahul13135@iiitd.ac.in
†IIIT Delhi, Email: sdeb@iiitd.ac.in
‡ST Microelectronics, Greater Noida, Uttar Pradesh, Email: fabio.carlucci@st.com

Abstract—**State of the art automotive microcontrollers (MCUs) implementing complex system-on-chip (SoC) architectures requires often additional functional patterns to achieve high degree of reliability. Functional pattern family includes test patterns checking internal device functionality under nominal condition. The development of these patterns is required to augment structural tests to achieve high test coverage. Moreover, it should be planned to minimize test time since it has direct impact on time to market. This paper proposes a reusable and time-economized approach for functional test pattern development using Test Information Model (TIM) discussed in the paper. The flow is also automated to exclude the potential source of error in the pattern generation. In applying this methodology, there is approximately 60%-70% reduction in pattern development time as compared to conventional simulation based pattern development. Also, the results show the efficiency of this approach in terms of pattern reusability by further reducing man-hours of the test engineer.**

Keywords–STIL, ATE, Functional pattern, Time-to-Market, Cost of Test

I. Introduction

The development cycle time is continually decreasing to meet very aggressive time-to-market requirements, while the product quality is expected to reach levels currently only known for military or aerospace products [1]. All this has to be achieved while minimizing the total cost for a device, where test costs are contributing an increasing portion. Also, there is a necessity to meet schedules with less engineering staff and also to increase reuse in designs, code and tools. A crucial aspect in this area is the generation of test patterns which are used for different objectives including qualification, burn-in, and production test. The outcome of these tests plays a critical role in deeming the system's readiness for production. In contrast to other domains where the generation of scan-type patterns by an Automatic Test Pattern Generation (ATPG) tools like Mentor Graphics Tessent®, Synopsys TetraMax™ is considered sufficient, the automotive industry requires often additional functional patterns to reach the quality level expected by its customers. To achieve this level of quality or target test coverage, modern scan test methodology has to be combined with built-in-self-test (BIST) for array-type modules, such as RAMs, ROMs and embedded flash. Any type of circuitry not covered by either of these techniques has to be tested by additional functional patterns to address structural test coverage weaknesses.

Test patterns are developed to ensure the full functionality of the device to be delivered to the customer keeping in mind that each test pattern has a direct impact on the device manufacturing cost. All these patterns verifies that the device operates as intended under nominal conditions. They are usually designed to be first serially loaded into internal device volatile memory (load phase) and then fetched from it (execution phase). The time needed to the completion of the loading phase is considerably higher than the time needed for the execution of the pattern (ratio close to 9:1). A non-code optimized test pattern has a direct impact on both test time (more time needed to load it and execute it) and Automated Test Equipment (ATE) vector memory. Thus, the test pattern development process should be planned to:

- Minimize the test time and hence, test cost.
- Minimize ATE vector memory.

Historically, these patterns were handcrafted or derived from simulations with little communication between design and test/product engineering [1]. Moreover, the traditional approach had its own drawbacks such as long test to time and others listed in Section II. Standard Test Interface Language (STIL) based pattern generation was introduced to remove pattern generation dependency on simulation [6]. However, writing STIL pattern manually can be time consuming and can introduce potential source of errors. To overcome such limitations, this paper proposes an automated flow that aims to generate these functional patterns in an industry accepted standard STIL (IEEE Std. 1450) [7]. The flow also deals with reusability of the test case for an Intellectual Property (IP) in different products. The aim is to reduce the test effort for same test in different verification environment. Test patterns are developed for IPs such as Phase Locked Loop (PLL), Analog to Digital Convertor (ADC), Digital to Analog Convertor (DAC), Thermal Sensors etc. to verify their intended functionality by validating the test cases in system level environment.

The paper sequence is organized in the following way. Section II explains the previously adopted simulation based pattern generation methodology, its limitations and shift to STIL based pattern generation. Section III discusses the proposed flow for pattern generation. This flow is practically implemented on different products, results of which are shown in Section IV. The paper is concluded in section V. Finally, the Section VI mentions the future enhancements of this flow.

978-1-4799-1743-3/15/$31.00 ©2015 IEEE

II. CONVENTIONAL FLOW OF PATTERN GENERATION

A. Simulation Based Pattern Generation

A conventional pattern generation flow is shown in the Figure 1. This flow starts with simulating the device behavior using an input stimulus usually derived from the Verification Database. This database consists of testbench environment and test cases that acts as input stimuli for the Design Under Test (DUT) [1]. These test cases are written in System Verilog or System C and are simulated using simulation tools whose output is a Value Change Dump (VCD) file that depicts the signal changes in the form of time related events. This event-based VCD file is translated to tester specific cycle-based force and expect values using some cyclization tools. Cyclization involves identifying the fastest signal period in the VCD, usually a clock signal, and using its period as the cyclized period. This cyclized period is used for all signals in the VCD to process all the event states and create drive/receive edge timings for all signals. These timings are all relative to the cyclized period. This ATE format file is run on silicon and in case of failures, the VCD for the specific failure needs to be regenerated. This take considerable time since changes must be fedback into the Verification Database and the entire flow has to be repeated every time a pattern update is necessary.

1) Limitation of this approach:

- $PatternVerification$: Any change to the test patterns-regardless of how simple-may necessitate verification. For each pattern, code compilation takes minutes to hours while simulation time takes hours to days thus affecting Cost of Test.
- $LargeVCDfiles$: VCD formats have been uniformly adopted way of capturing simulation output. Every ATE has limitation like vector memory, number of permitted drive and receive edges.VCD files can grow quite large for larger designs, or even for long simulation time in case of smaller designs.
- ATE debug time can increase exponentially due to re-simulation, re-conversion, and more ATE debug activities if VCDs are not created and converted properly.

B. STIL Based Pattern Generation

Reducing Cost of Test (CoT) is being more demanding which requires reducing test time to meet aggressive Time to Market (TTM). CoT and TTM can be hindered by factors like long vector simulation times, large VCD files, VCD to ATE binary conversion, ATE test time per hour cost. Thus, there is a need to remove pattern generation dependency on simulation.

STIL based pattern generation eliminates translation into tester specific formats by establishing direct link between ATPG tools and ATE [6]. ATPG tools like Mentor Graphics Tessent®, Synopsys TetraMax™ typically provide scan test pattern in STIL format or BSD Compiler from Synopsys provides Boundary Scan pattern which requires manual handling and knowledge of the standard to generate the test pattern. A suitable method for verifying the test patterns is achieved by creating a relatively simple testbench that accesses the patterns directly during simulation through a Verilog Programming Language Interface (PLI) [8]. Such a method avoids creating a large testbench which includes all the test vector. Using this method, test data can be modified without recompiling the design. However, manually writing a STIL pattern can be time consuming and error prone since pattern has to be written vector to vector. This may lead to larger time to test thus affecting time to market.

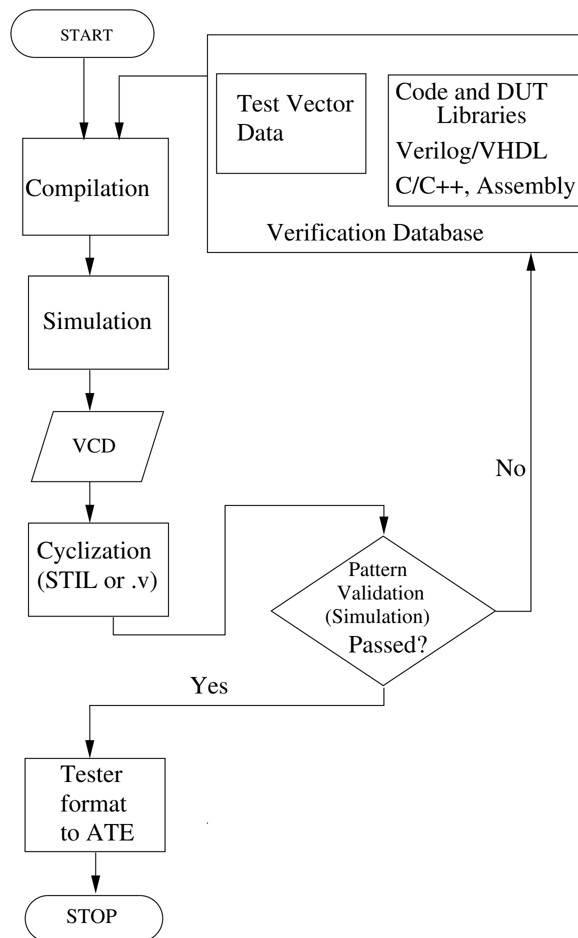


Fig. 1: Simulation Based Pattern Generation Flow

III. PROPOSED FLOW OF PATTERN GENERATION

The proposed approach for pattern generation has been shown in Figure 2. A library of STIL procedures or templates are created that are specific to IPs whose pattern are created. This is a one time investment since these templates can be reused for the same IP in a different SoC. A database is also created containing pattern specific information. Using these two information, an ATE format pattern i.e STIL pattern is created. This pattern is then validated by simulation. The main focus of dynamic validation is to verify whether the patterns are functionally right (produce the correct and desired results) and do not harm the device or the ATE. The dynamic validation can only be performed through simulation. The ability to simulate the vectors in ATE formats (like STIL) flags off many tester failures early and reduces cost of tester debug. If the pattern fails the schema can be manually edited for debug purposes. This speeds up debug and reduces pattern

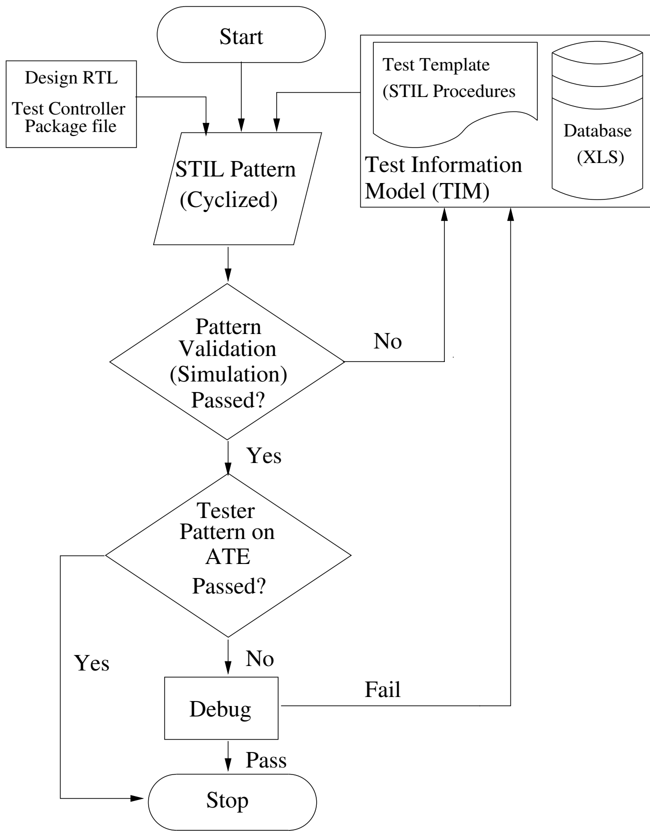


Fig. 2: Proposed Pattern Generation Flow

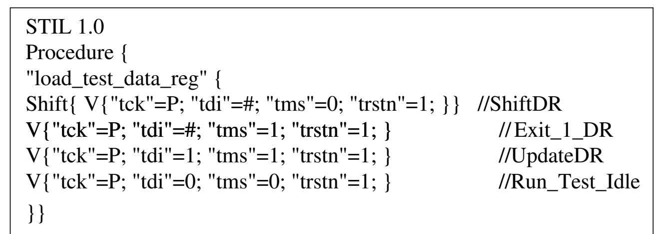
regeneration time as simulation is not required to generate VCD file.

A. *Test Information Model*

Test Information Model (TIM) contains database used to convey information about the test framework of an SoC or embedded core as well as to describe complete test programs that it can execute. TIMs serve the purpose of delivering test vectors generated in ATE format (STIL) to test engineering responsible for running the tests on ATEs. TIM is composed of :

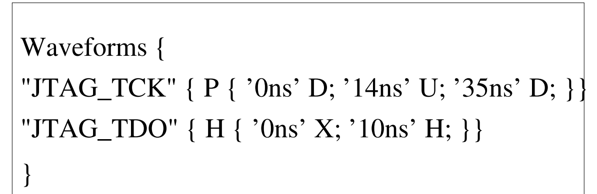
1) STIL Library: This is composed of a set of STIL Procedures specific to IPs as well as procedures commonly used in different SoCs to configure their test data registers (TDRs). A STIL procedure is a set of patterns to be executed in an order to accomplish a task like precondition (load) the scan chain, and to observe (unload) the scan chain. The tool will make use of this library to generate a STIL file for a particular IP in a SoC. With a new IP to be functionally tested, a generic procedure to verify its functionality can be created which can be reused in different SoCs. Library of STIL procedures contains:

- Generic STIL procedures: Ex. *load_test_data_reg*–This includes sequence of patterns to configure the test data registers as shown in Figure 3 with Test Access Port (TAP) states. These can be reused across SoCs even with different functional blocks but with same test controller.
- Specific STIL Procedures: These are specific to a functional block. For example, *program_fuse, read_fuse* to write and read memory locations respectively of One Time Programmable (OTP) fuses. These can be reused for SoCs with common functional blocks.



```
STIL 1.0
Procedure {
"load_test_data_reg" {
Shift{ V{"tck"=P; "tdi"=#; "tms"=0; "trstn"=1; }}   //ShiftDR
V{"tck"=P; "tdi"=#; "tms"=1; "trstn"=1; }            //Exit_1_DR
V{"tck"=P; "tdi"=1; "tms"=1; "trstn"=1; }            //UpdateDR
V{"tck"=P; "tdi"=0; "tms"=0; "trstn"=1; }            //Run_Test_Idle
}}
```

Fig. 3: An abstract of STIL Library



```
Waveforms {
"JTAG_TCK" { P { '0ns' D; '14ns' U; '35ns' D; }}
"JTAG_TDO" { H { '0ns' X; '10ns' H; }}
}
```

Fig. 4: Waveform Construct

2) Database: This is a small schema created for every pattern containing information about signals that are forced or expected in that test case. Table I is an abstract from such a database showing an example that defines a Clock along with its timing and an Output to strobe at the specified tester cycle instance. Using this information, the script in the tool will generate the Waveform construct of STIL pattern as shown in Figure 4. All the timing information is with respect to the tester cycle period again specified in the database.

TABLE I: DATABASE ABSTRACT

Clock	Default State	T1	T2
JTAG_TCK	D	14ns	35ns
Output	Default State	Strobe Time	-
JTAG_TDO	X	10ns	-

B. *An Automated Solution*

Lack of automation in functional test pattern generation and long test development time have been cited as one of the reasons to move away from function testing to structural testing. However, functional tests are more effective in testing at speed, under realistic electrical noise [10]. Automation offers potential for both reducing test engineer workload during pattern development and risk of error. The source of automation adopted is scripting in Perl. The script has the STIL rules intelligence embed into it. DUT Register Transfer Language (RTL) in Verilog Hardware Description Language (HDL) is used to extract the top level pin information along

```
Pattern "_pattern_" {
    W "_default_WFT_";
Ann {*PLL is powered down and configuration is done*}
Call "load_test_data_reg" {"tdi"=11000111001100010  ; }

Ann {*PLL should be in power down for minimum 10us*}
Loop 300 { V { "tck"=P; "tdi"=0; "tms"=0; "trstn"=1; "system_clk"=P; }}

Ann {*PLL is  powered up*}
Call "load_test_data_reg" {"tdi"=11000111001100000   ; }

Ann {*Wait for PLL to lock*}
 Loop 20000 { V { "tck"=0; "system_clk"=P; }}
}
```

Fig. 5: Phase Locked Loop(PLL) functional test case

with their direction. This information is used to generate the STIL Signals construct. Test controller package file is utilized to obtain TDR bits information. This information is used as a reference to create functional test pattern from a sequence of call to *load_test_data_reg* procedures by programming appropriate bits of TDRs in database.

An excerpt from functional test pattern of PLL is shown in Figure 5 in which first call to *load_test_data_reg* procedure in Figure 3, power down the PLL and configure it. Then, second call to the procedure power up the PLL after a certain wait period. Lastly, wait period is given with system clock running to allow PLL lock signal to go high.

The flow makes use of this TIM to generate STIL pattern for every block that requires verifying its functionality. The STIL patterns obtained are in tester format which are first validated by simulation tools like Cadence NCSim or Synopsys VCS® and then delivered to the test engineering for running the test cases on an ATE.

C. Test Generation Efficiency

The idea of proposing this methodology at the outset was to reduce test time and keep manual intervention as minimum as possible to reduce the potential of human error. Also, expectation was to buildup a library of procedures to facilitate reusability. Thus, the test engineer only has to give a little information in database form. The proposed flow in addition to reduction in pattern development time, permits the generation activity to be initiated as soon as first verified RTL of the DUT is available.

D. Pattern Reuse

Signal Groups construct in STIL provide a mechanism to reference a signal using alias name. This property can be used to build up a library of STIL procedures with generic names. The tool will take care of assigning generic names to the actual signal names in the STIL pattern file by specifying it in the database. Thus, apart from Signals, Waveform constructs and power-up sequence, same pattern can be reused for an IP in a different SoC.

IV. RESULTS

The proposed approach of test pattern generation was used for analysis on SoC 1 and SoC 2. SoC1 is a processor with 270 memories, 4 PLLs, DDR PHY, 2 Thermal Sensors, a process monitor on 40nm technology with embedded MIPS core and close to 5.5 lacs flops. SoC2 on the other hand, has 3 PLLs, 110 memories, 10-bit SAR ADC, 18-bit MIC ADC, 18-bit Stereo ADC, Audio DAC with Cortex ARM processor having 4 lacs flops. Plan to test an IP is deduced from its functional specifications. However, IP reuse in a different product requires almost no time to develop its pattern. The functional patterns are generated in STIL format. These are validated by simulation using Cadence NCSim by creating a relatively simple testbench that instantiates the DUT and accesses the patterns directly during simulation through a Verilog PLI.

To accept the benefits of the proposed approach, it is appropriate to not only compare the effort spent for pattern development, but also to consider reuse aspects. To demonstrate these capabilities, the typical effort spent for the tester pattern develoment process is compared with the traditional approach especially for the benefits in case of device families. The comparison of the two approaches has been shown in Table II in the form of pattern generation time.

TABLE II: COMPARISON OF TRADITIONAL AND PROPOSED APPROACH

Product (SoC)	Number of Patterns	Pattern Generation Time(Weeks)	
		Traditional Approach	Proposed Approach
SoC1	15	8-10	2-3
SoC2	10	6-8	1.5-2

Due to development of library of STIL procedures for IPs, the advantage of pattern reuse can be exploited as can be inferred from the Table III. Even more significant, when it comes to generating patterns for a product family (having slightly differrent architectures), the pattern generation shows the real power of reuse as compared to hand edits.

TABLE III: PATTERN RE-USE RESULTS FOR SoCs

Product (SoC)	Common IPs	First-time Pattern Development Time(Man-Hours)	Pattern Reuse Time(Man-Hours)
SoC1 & SoC2	OTP Fuse,PLL, Compensation Cell, ADC, DAC	10-15	2-4

V. CONCLUSION

As can be implied from the results, test pattern generation using the proposed flow reduces the test development time by approximately 60%-70% and thus, test cost. Also, test patterns are generated in ATE format (STIL) which can be delivered directly to test engineering. It, thus eliminates waiting for slow gate level simulations to generate VCD file and use of cyclization tool to convert VCD file into ATE format.

Moreover, this flow allows to create test patterns as soon as first verified RTL is available rather than waiting for the gate level design(usually available late) in conventional approach or for ATPG tools to dump the initial STIL template. This results in quicker silicon bring-up.

VI. FUTURE WORK

The flow is limited to a set of devices having common test architecture. There is a scope to support a wider range of products with different test architectures as well thereby making it more generic. Moreover, with new IPs to be functionally tested, STIL procedure library has to be continuously updated to make this approach reusable for them.

VII. ACKNOWLEDGEMENT

This work is partially supported by the DST INSPIRE Faculty Fellowship granted by the Department of Science and Technology, Govt. of India.

REFERENCES

[1] Ernst Aderholz, Heiko Ahrens, Michael Rohleder, "Bridging the gap between Design and Test Engineering for Functional Pattern Development", *Int. Test Conf.*, pp. 1-10, 2008.

[2] Dr. Anke Drappa, Peter Huber, Jon Vollmar, "Automated test program generation for automotive devices", *Int. Test Conf.*, pp. 1-10, 2010.

[3] Prokash Ghosh, Celia John, Ajay Gupta, Veerabhadrarao Siripurapu, "Enhancement of Production Pattern Development Methodology and Best Practices", *Int. Symp. on Electronics System Design*, pp. 153-157, 2013

[4] P. C. Maxwell, I. Hartanto and L. Bentz, Comparing Functional and Structural Tests, *in Proc. Int. Test Conf.*, pp. 400-407, 2000.

[5] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed-Signal VLSI Circuits*, Boston; Kluwer Academic Publishers, 2000.

[6] H. Lang, B. Pande, H. Ahrens, "Automating test program generation in STIL - expectations and experiences using IEEE 1450 [standard test interface language]", *IEEE European Test Workshop*, pp. 99-104 , 2003.

[7] IEEE Standards Association, *Standard Test Interface Language (STIL) for Digital Test Vectors*, IEEE Std.1450-1999.

[8] N. Nemati, M. Namaki-Shoushtari, Z. Navabi, "A mixed HDL/PLI test package", *Design & Test Symposium*, pp. 518 - 523, 2010.

[9] J. Hudec, "Some results in automatic functional test design for VLSI circuits", *32nd International Conference on Information Technology Interfaces (ITI)*, pp. 635-638, 2010.

[10] S. Kundu, T.M. Mak, R. Galivanche, "Trends in manufacturing test methods and their implications", *International Test Conference*, pp. 678-687, 2004.

[11] Hau Lam, "New design-to-test software strategies accelerate time-to-market", *International Electronics Manufacturing Technology Symposium*, pp.140-143, 2004