Automated System Level Functional Test Program Generation on ATE from EDA using Functional Test Abstraction

Motoo Ueda, Shinichi Ishikawa, Masaru Goishi, Satoru Kitagawa, Hiroshi Araki, Shuichi Inage

ADVANTEST Corporation
336-1, Ohwa, Meiwa-machi, Ora-gun, Gunma 370-0718 Japan
motoo.ueda@jp.advantest.com

Abstract

This paper introduces new capability on System on a Chip (SoC) ATE, called "Functional Test Abstraction (FTA)", which allows us to execute an automatically generated system level functional test program from the system level design verification environment. The device under verification and device under test can be a complex SoC which has multiple logic time domains and multiple interfaces of the same or different types.

1. Introduction

In recent years the number of time domains in LSI chips has increased rapidly and a variety of new interfaces have been introduced. In the design verification environment, system level verification is used to verify an entire system using a highly abstracted verification language.

We have developed a new logic test capability for SoC test systems, which can execute a system level functional test that is equivalent to system level verification. The capability shown in Figure 1, includes Packet Sequencers [1], which can execute a highly abstracted functional test program, a synchronization system to control the synchronization between the different time domains of the Packet Sequencers, and a software environment that uses a programming language such as Verilog and does not modify the original verification program.

In addition to that, we have implemented a tester model in the design verification environment, which can be used as a transactor for interfaces. It allows us to evaluate the device under verification (DUV) using tester specifications and limitations without additional tester specific development in the tester environment. Figure 2 shows that environment. The tester model is called FTA-TBLib.

We have named the entire capability Functional Test Abstraction (FTA).[2]-[4] It is based on the idea of Protocol Aware testing [5]-[9], which was introduced by Broadcom. The basic idea of Protocol Aware testing is that

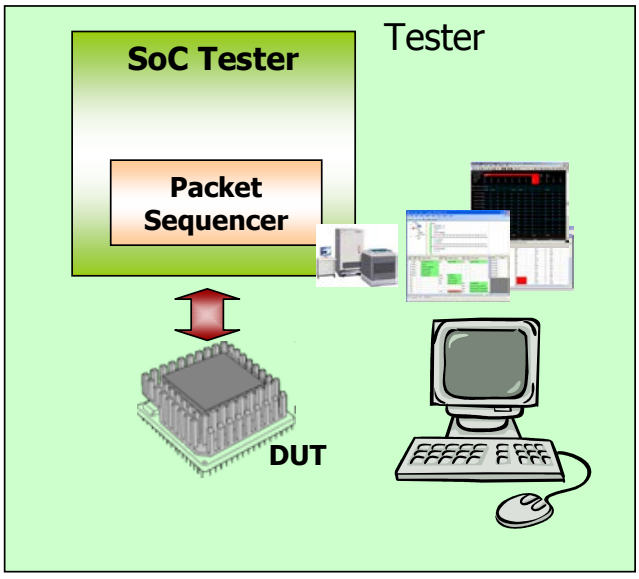

Fig. 1 FTA (Functional Test Abstraction) on ATE

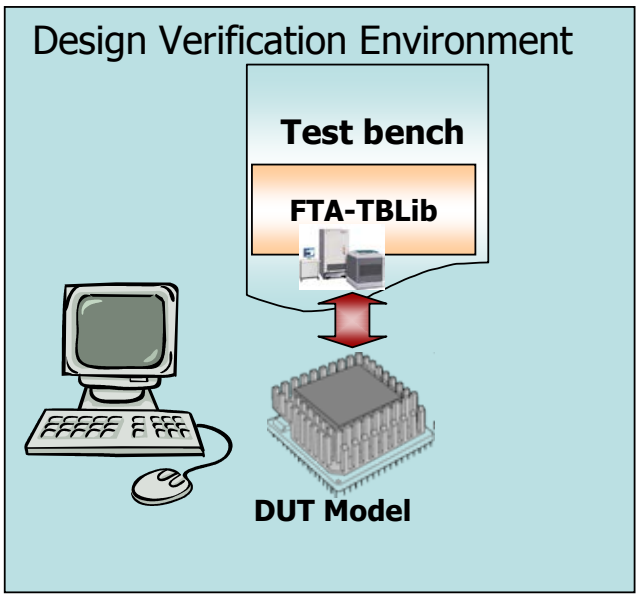

Fig. 2 Test Bench with FTA-TBLib

978-1-4673-1595-1/12/$31.00 ©2012 IEEE

the ATE knows the protocol of the interfaces and the ATE customer can program using abstracted transaction level program interfaces instead of the "1's and "0's" of logical test vectors. I would like to say that FTA is an implementation of Protocol Aware testing capabilities.

At the end of the paper we tell how we have demonstrated these capabilities using an actual test device of our own design as the DUV and device under test (DUT).

2. Problems in legacy logic test capability in SoC ATE

In recent years, multiple time domain capability for ATE digital test has become popular. However automatic test program generation from electronic design automation (EDA) using multiple time domain capability has a number of issues to be solved. For that reason, complex functional test, in many cases, is not executed on ATE.

System level test, using a mother board, which has the peripheral devices of the final application, is executed for production test as the alternative solution. However, this solution makes it difficult to implement a flexible failure analysis capability when defects occur, so it takes a very long time to discover the cause of a defect.

The issues of using legacy ATE cycle based logic vector generators are discussed in the sub-sections below.

2.1 Differences in abstraction level

In the high level verification language, the test scenario calls a transactor function, which is the verification model of the interface, with a transaction value as an argument, such as an address and/or data. The transactor converts the transaction value to a physical bit stream. On the other hand, the legacy logic vector generator on ATE can only handle a low level "1s" and "0s" logical signal. Software conversion is necessary to generate an ATE program from EDA. The problem is that sometimes the bit stream is not deterministic even if the transaction value is deterministic, because sometimes a random function is used or the order of the packets is not deterministic due to the condition of the DUT. In that case deterministic software translation is not applicable.

2.2 Correlation from EDA to ATE

There are fewer limitations in the design verification environment compared with ATE. ATE has limitations such as hardware latency, maximum data rate, timing edge placement, number of edges per cycle, timing accuracy and so on. The value change dump (VCD) and cyclization method is usually used to convert test vectors for legacy ATE. Some existing logic test ATE systems have an event-based architecture [10], which can eliminate the cyclization process required by cycle based legacy ATE, and the limitations due to cycle based timing inflexibility can be relaxed. Of course, with both of these ATE architectures, design verification must be done on the condition that output vectors can be deterministic. The information in the generated vectors does not contain ATE limitations. Usually after the vectors are generated, violations due to ATE limitation must be checked using tools such as a virtual tester. This causes a turnaround time issue when translating from EDA to ATE.

2.3 Hand-shaking between other time-domains

In system level design verification with multiple interfaces on the DUV, each test scenario for a particular time domain is individually written. To complete the entire verification sequence, one interface may require notification from another interface to decide when to go on to the next step. Legacy ATE has mechanisms to synchronize between time domains, but, in general, automatic test program generation with this type of synchronization for system level verification has not been realized.

2.4 Branch Syntax conversions

In the system level design environment, branching syntax can be written to control the test flow or to determine the next data input to a DUV based on a previous test result or a condition of the DUV. To do the equivalent branching on legacy ATE, the test control CPU must be used. In many cases the branching cannot be executed within the required test sequence latency.

2.5 Independent input / output timing

For many packet based interfaces, the input and output are independent. Even if the data rate is exactly same, the communication timing is asynchronous. The stimulus pattern and expected pattern of legacy ATE vector sequencers are stored at the same vector memory address. This means that input and output signal timing must always be synchronous. On the other hand, the design verification environment does not have such a restriction. When we convert from EDA to ATE, the simulation scenario must be limited to be synchronous between output and input. This is not realistic for system level functional test.

2.6 Real-Time hand-shaking between ATE and DUT

An interface which has a complicated protocol and a state machine requires real time handshaking, such as acknowledgement/negative-acknowledgement (ACK/NACK), to transfer knowledge of the interface state across the interface, or to initialize the interface while controlling and monitoring the condition of the state machine. The VCD timing chart which is translated from EDA is just one example of this. In the real world, multiple kinds of bit streams are generated by the DUT. Legacy vector sequencers have match loop capability to handle hand shaking with the DUT. It is possible to wait for a phase-locked-loop (PLL) to lock, for example. However, it is not enough to only support an ACK/NACK level of handshaking complexity. It is necessary to have "if then else" branching capabilities as well.

3. Implementation of FTA

The hardware and software components developed for FTA are described in this section.

3.1 Packet Sequencer

Figure 3 shows the block diagram of the Packet Sequencer. This is distinct from a legacy vector sequencer. The Packet Sequencer has two independent sequencers, one for the driver and one for the comparator, which are controlled by the common sequencer. The two independent sequencers can operate synchronously or asynchronously. The Packet Sequencer has a micro program control architecture that is similar to a general purpose digital signal processor (DSP). The sequencer has a 32bit internal bus and can handle 1 to 32bit parallel data. The base operation frequency is 250MHz. When performing 32bit parallel operations, the sequencer can handle an 8Gbps data stream, since 0.25Gbps x 32=8Gbps. There are also hardware components for real time operation that are controlled by the sequencers. These can realize packet signal generation and detection for interface standards such as PCI Express.

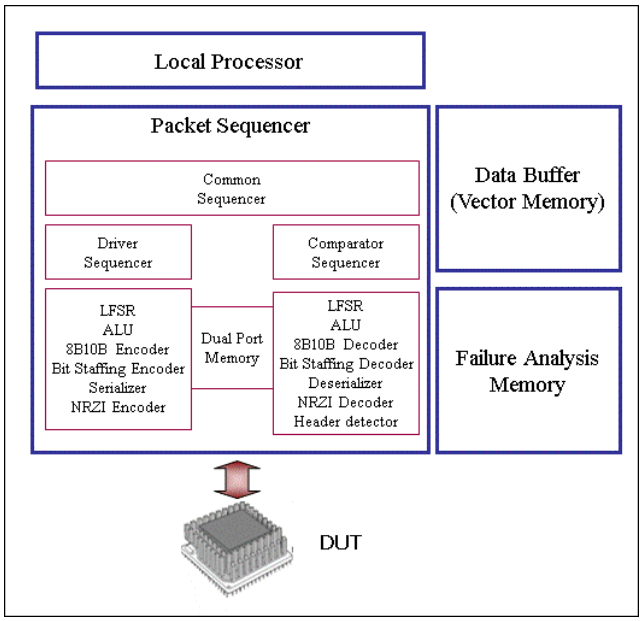

Fig. 3 Block Diagram of Packet Sequencer

We have implemented the Packet Sequencer hardware components described below.

1. Linear feedback shift register (LFSR) (cyclic redundancy check (CRC) generator/detector, scrambler, de-scrambler, random data generator/detector)
2. Arithmetic logic unit (ALU)
3. 8b10b encoder/decoder
4. Bit stuffing encoder/decoder
5. Serializer / deserializer
6. Dual port dcratch pad memory
7. Header detector
8. Non return to zero, inverted (NRZI) encoder / decoder

These components are designed to be as programmable as possible to support custom or special protocols.

3.2 Synchronization hardware

We have developed a low latency synchronization system to work synchronously or asynchronously between different time domains. Both transmit and receive signals are connected in a star structure to minimize latency and increase stability. In this concept, multiple time domains can work in parallel with low latency and superior stability. Also a local processor is implemented to execute branching syntax, such as "while" or "if", without interaction with the tester controller CPU. This is implemented above each Packet Sequencer. Each local processor can access registers from the different time-domains to avoid the (otherwise) necessity of inter communication between local processors. This allows us to execute branching syntax with low latency and good stability.

3.3 Test Program Language

We have developed an FTA procedural programming meta language to realize a highly abstracted system level functional test program. The FTA meta language must have the same capabilities as high level design verification

languages. This allows us to support different kinds of verification languages using a line-by-line conversion tool. Some basic concepts of the FTA meta language are listed below.

1. Separate the test scenario, pay load data, and protocol definition.
2. Enable a coder / decoder description to support many protocols with minor adjustments.
3. In the test scenario, enable the genceration of stimulus and response data using a procedural programing method.

3.4 Debug tools

We have created graphical user interface (GUI) tools to enable debugging at different abstraction levels. Debug tools for three levels of abstraction are described below.

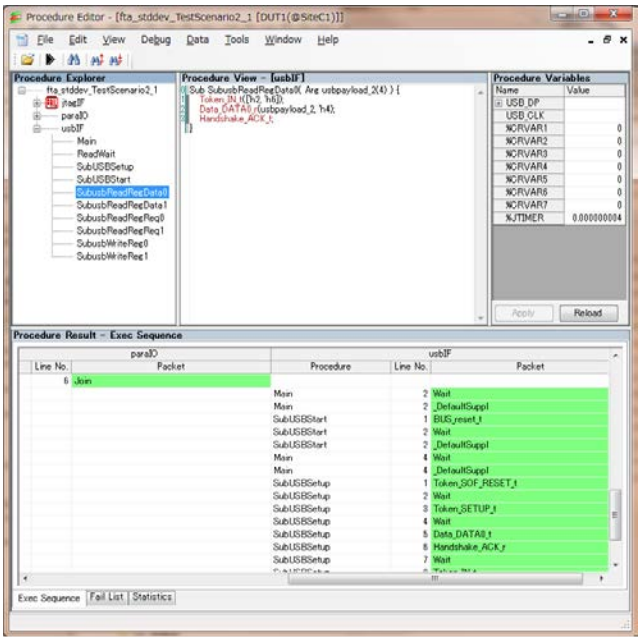

Fig. 4 Procedure Editor
Test Scenario Level

1. Figure 4 shows a source code trace of a highly abstracted test scenario language at the test scenario level
2. Figure 5 shows the order of packet generation and received packets at the packet level.
3. Figure 6 shows the timing chart at the bit stream level

3.5 FTA verification library

We have created an FTA model library which is called FTA-TBLib. FTA-TBLib contains information about the hardware limitation of FTA on our ATE, such as hardware latency, timing restrictions, and so on. When design verification is done using FTA-TBLib for the verification library, the result can include the ATE hardware restrictions. This means the verification result is the same as virtual test. We expect that the time from design to production can be dramatically improved.

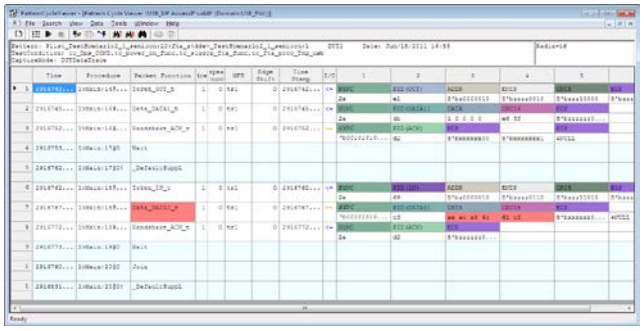

Fig. 5 Pattern Cycle Viewer
Packet Level

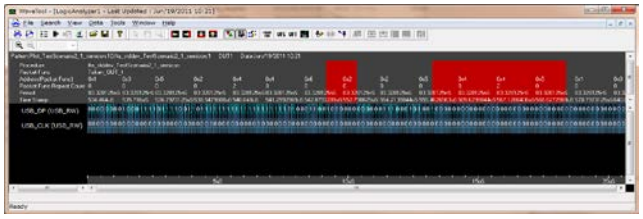

Fig. 6 Wave Tool
Bit Stream Level

3.6 Conversion tool

We have developed a conversion tool to convert from a test scenario on the test bench to the FTA meta language. The abstraction level of both languages is similar and almost line-by-line conversion is possible, making it easy for the customer to understand the meta language. We have developed a conversion tool from Verilog to FTA as our first step.

4 Experimental Result

4.1 DUV/DUT Design

We have developed a DUT, shown in Figure 7, to assess the FTA capabilities. The DUT has a joint test action group (JTAG) port, a parallel port, and a universal serial bus (USB) port as external interfaces. A microprocessor and common memory are inside the field programmable gate array (FPGA). It is possible to access the internal common memory through each interface asynchronously. The DUV/DUT is designed using an FPGA and a USB

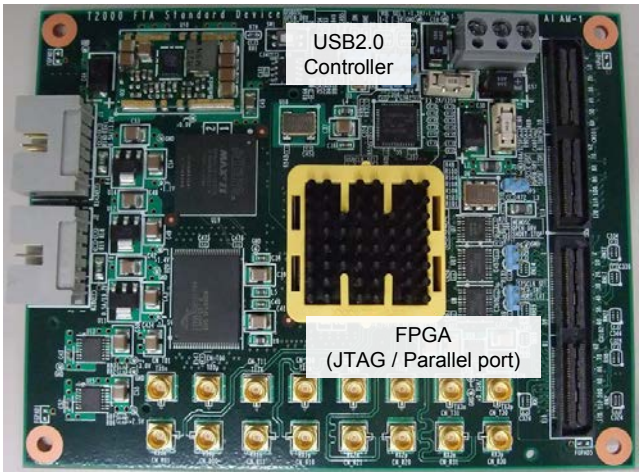

Fig. 7 DUT

micro controller for demonstrating the FTA capability. The FPGA code is written in Verilog language. A commercial USB verification IP model from an EDA vendor is used as the USB micro controller design model, since the USB micro controller is outside of the FPGA and it is a commercial device. The entire DUV/DUT simulation model is realized by using these two models.

Figure 8 shows the ATE Packet Sequencer to DUT connections for our test.

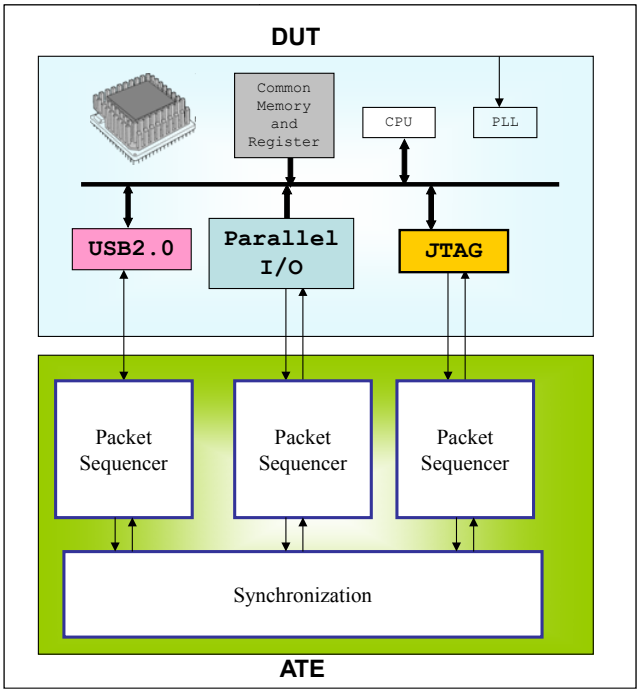

Fig. 8 Block diagram of DUT and ATE connection

4.2 Design Verification and Test scenario

Figure 9 shows the test scenario for our examination. The test scenario is written in the Test Bench as shown below.

1. Reset DUT through JTAG port
2. Initialize USB port
3. Write data to common memory through USB port
4. Read-modify-write common memory, access through JTAG port
5. Read data from common memory through parallel port
6. Read data through USB port

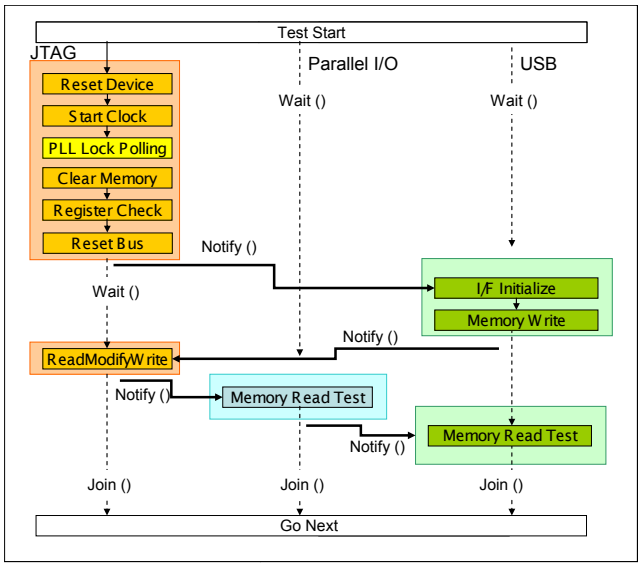

Fig. 9 Test Scenario
Functional test with three time domains

Design verification executed the scenario above on the test bench. The DUV model and the transactor are connected using FTA-TBLib. We successfully got a pass result on the test bench.

4.3 Program generation

The test program was automatically generated from the test scenario using our conversion tool. Also, a loadable object program was generated by the FTA meta language compiler.

4.4 Program execution on ATE

This scenario has three different time domains which operate asynchronously. The three Packet Sequencers also operate individually. The sequencers send notification signals to other sequencers or wait for and receive

notification signals from another sequencer through the synchronization system. Thus, message based sequencing control between time domains is accomplished. The common memory bus inside of the DUT is assigned to a particular interface port during access. At that time other interface ports must wait until the access is completed.

For the read-modify-write, read data from the common memory is modified using an arithmetic or logical calculation. To do that on legacy hardware, the read data must be transferred to the tester controller and the modified data must be written to vector memory, and then the memory data in the DUT must be updated. When the Packet Sequencer is used, data from the DUT is stored in dual port scratch pad memory, the modification operation is executed using the ALU, and then the modified data is stored in the DUT. This is very fast and easy to convert from the design verification scenario.

During data communication using USB, a hand shaking communication protocol is required. When the ATE is the host device, the transaction below is necessary to read data from the DUT.

a. Data transfer request from ATE to DUT

b. Data transfer from DUT to ATE

c. Acknowledge notification from ATE to DUT

After the ATE sends the transfer request to the DUT, the ATE waits for the data transfer from the DUT. The data transfer timing from the DUT is non-deterministic because it depends on the status of the DUT. To correctly receive the data, the Packet Sequencer starts to capture the data after waiting for the packet header code using the header detector. After the data is correctly received, the ATE sends an acknowledge packet. The transferring side waits until the data receive is complete.

Using this example, we demonstrated that asynchronous and non-deterministic timing operations between transmit and receive on the Packet Sequencer were working correctly.

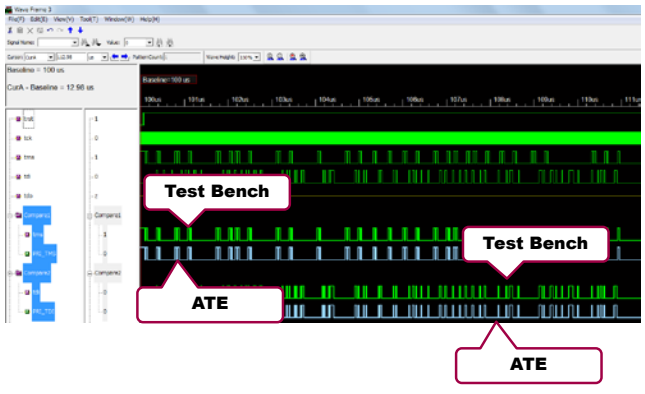

Fig. 10 Timing chart comparison between EDA and FTA on ATE

4.5 Executed result

Figure 10 shows a comparison between the design verification result and on-line execution on the ATE using FTA. There is complete correspondence between the test bench and the ATE. Figure 11 shows that the test sequence is correctly controlled by the ATE. The time stamp and procedure name of each executed procedure in the test scenario are displayed in the Procedure Editor tool. Part of the test scenario of Figure 9 is displayed, starting from within the JTAG read-modify-write, then the Parallel I/O memory read test, and then the USB memory read write test. These results demonstrate the concurrence of the design verification result and the ATE on-line test result, running with an automatically generated program using our own designed test device.

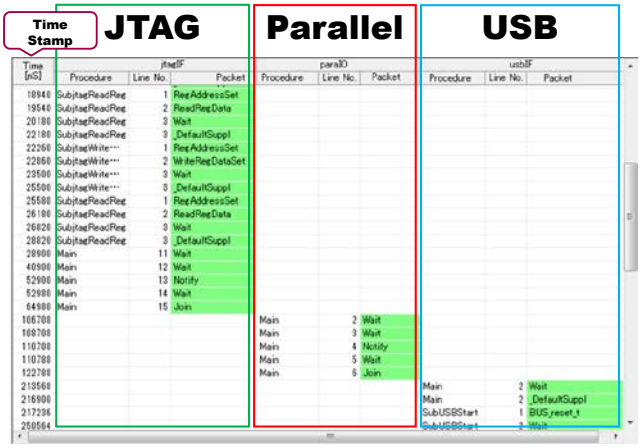

Fig. 11 Monitor of test sequence on ATE (FTA)

5. Limitations and Next Step

The capabilities on this paper have been implemented on our commercial ATE, and we are gathering feedback from customers. The FTA capability on our platform does have some limitations, such as a particular handshaking latency. When necessary, we are asking our customers to implement longer latency handshaking capabilities as DFT. In parallel, we are planning to improve the FTA capability itself, according to the customer feedback.

6. Conclusions

The goal of FTA is completely automatic program generation for system level functional test under multiple time domains. We have demonstrated the entire flow from design to test using our own devices. We will engage with our customers and improve this capability as our next step.

7. Acknowledgements

We would like to thank the huge number of R&D members outside of the authors who were involved in FTA development and the top management of ADVANTEST, who provided the big investment.

8. References

[1] Masaru Goishi, Hiroyasu Nakayama, Masaru Tsuto, "Test apparatus and test method", US Patent 8059547
[2] A.T Sivaram, "Functional Test Abstraction", ITC 2010 Poster 12
[3] Satoru Kitagawa, "SoC Test System Architecture Corresponding to New Test Methodologies", SEMI Technology Symposium 2010
[4] Shinichi Ishikawa, "Test apparatus and test method", US Patent 8060333
[5] Andrew C. Evans, "The New ATE: Protocol Aware", ITC2007, paper 20.1
[6] Eric Larson, VLSI Test Symposium 2007, "Inovative Practices"
[7] Eric Larson, "Protocol Aware ATE", 2008 Beijing Advanced Semiconductor Technology Symposium
[8] V. Aggarwal, "Protocol Aware ATE with FPGA-based Hardware", AUTOTESTCON, 2008 IEEE
[9] Richard Rovbbers, "Trends in Test part 5 – Protocol Aware Test", EMT Worldwide June 09, 2009
[10] Jerry Katz, Rochit Rajsuman, "A New Paradigm In Test For The Next Millenium", ITC2000, paper 18.1