Automated Flow for Test Pattern Creation for IPs in SoC

D. Sargsyan, G. Harutyunyan, S. Shoukourian, Y. Zorian
Synopsys
{david.sargsyan, gurgen.harutyunyan, samvel.shoukourian, yervant.zorian}@synopsys.com

Abstract

With the increasing technological complexity modern SoC designs continue to grow in size and involve increasingly more IPs. Therefore, it becomes much harder to complete testing of large SoCs within the desired schedule and cost. Usually an automated hierarchical test helps to solve this problem efficiently but for such systems the preparation of input data, especially IP level information and description of test patterns, usually takes very long time. In this paper, an efficient solution for preparing input data for hierarchical system is presented.

1. Introduction

The embedded test solutions for system-on-chips (SoCs) developed a few years ago are becoming non-sufficient for nowadays designs since newer designs are much bigger, faster, hierarchical and much more sensitive to area, timing and power. For example, test solutions developed for 45nm or 28nm technology nodes will not provide the same level of test quality for 16nm or 7nm technology nodes, as defects and failure mechanisms change with process technologies shrink.

Usually, different approaches and standards are used for IP integration into SoC. At the chip level, the total number of test channels is limited such that all core-level test channels cannot be accessed at the same time.

In [1], authors proposed SoC testing with channel sharing/broadcasting methodology. This solution is good for medium-sized designs that can still run ATPG at the chip level.

In [2], authors presented hybrid test methodology incorporating modular test and hierarchical test. A case study is illustrated to compare different test flows used in the EDT (Embedded Deterministic Test) compression based SoC testing context.

In general, hierarchical test gives designers flexibility to schedule test of individual interface IP blocks and other cores for parallel and serial testing to optimize test time and power consumption during test [3], [4]. The flexible test schedule can significantly reduce test time, especially for designs with a large number of high-speed I/Os.

Due to increase of SoC complexities, IP level integration and test pattern porting from IP level into SoC level cannot be done manually anymore and efficient automation techniques are needed to meet time-to-market requirements. In this paper, automated solution for generating IP level information for hierarchical test is presented.

The suggested approach is implemented in Synopsys DesignWare STAR Hierarchical System and was successfully applied to a comprehensive set of IPs (including DDR3, DDR4, LPDDR4, HBM, USB2, USB3, PCIe2, PCIe3, SATA, HDMI, etc.).

2. Hierarchical test system

As mentioned above, our experiments were done on STAR Hierarchical System (see Fig. 1), where:

1. The input data for the system is information about IPs and description of test patterns, which is described using proprietary language called MASIS.
2. IPs are wrapped by IEEE 1500 compliant wrappers.
3. IP level test patterns are automatically ported to SoC level controlled by Server which is IEEE 1149.1 [5] and 1687 [6] compliant.

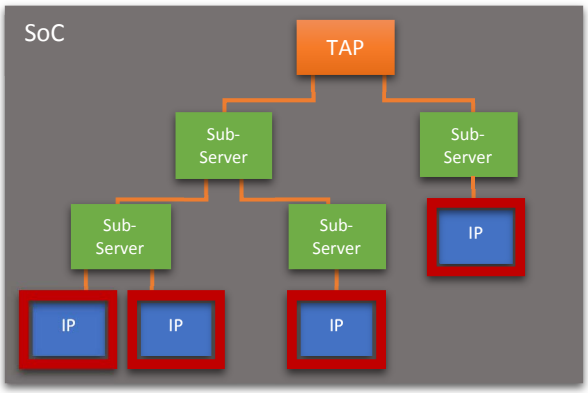

Fig. 1. Hierarchical test architecture for SoC

978-1-5386-3299-4/17/$31.00 ©2017 IEEE

The IEEE 1500 hardware architecture is comprised of an Instruction Register (the Wrapper Instruction Register), and two data registers, the Wrapper Bypass Register (WBY) and the Wrapper Boundary Register (WBR). The use of Core Data Registers (CDRs) is also anticipated by the standard. Access to these registers is provided via a set of wrapper interface ports. Fig. 2 displays the architecture of IEEE 1500 standard [7].

There are two categories of wrapper interface ports:

- Wrapper Serial Ports. (for serial access to the wrapper);
- Wrapper Parallel Ports (for parallel access to the wrapper).

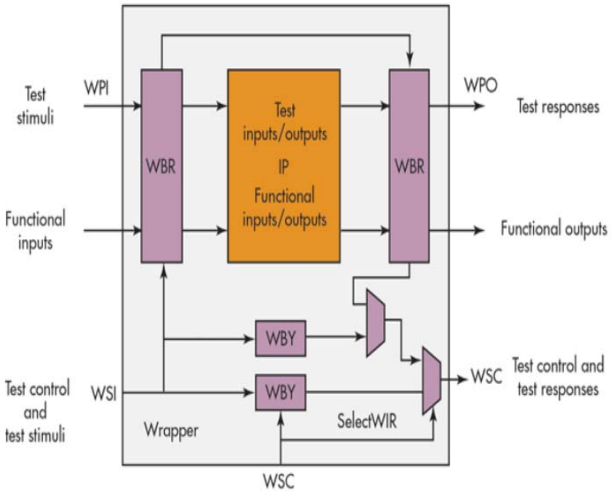

Fig. 2. Architecture of IEEE 1500

Thus, using IEEE 1500 with IEEE 1149.1 and 1687 standards allows to have a flexible hierarchical test solution for complex SoCs.

3. Test pattern description language

MASIS provides set of parameters which are necessary to describe the structure of a given IP. It has the following sections:

- Port description – Name of ports and attributes (function, direction, range, etc.);
- Core internal serial test data – CDR and its attributes;
- Isolation chain – WBR or CDR ordering;
- WDR – Wrapper Data Register (WDR) chain;
- Comments – line (//…) and block (/* … */) comments are supported.

Some examples of format for ports and test patterns are adduced below:

```
Port {
 clk_1 {
  PortInfo = "Clk from PLL"
  Direction = Input
  Tag = Clock
  MinFreq = 50.0
  MaxFreq = 150.0
 }
 din {
  PortInfo = "Input data"
  Range = "[7:0]"
  ExpandFactor = 1
  Direction = "Input"
  Tag = Static
  IsolationCell = DynamicTestIn
 }
}
```

MPL (MASIS Pattern Language) is used to describe test patterns with high level description. MPL provides:

- Verilog task-like test pattern description flow;
- Supporting of binary, decimal, octal, hexadecimal and string type operands;
- Support of expressions (concatenation, replication and grouping);
- Support of bit-select and range-select;
- Support of comments;
- Tcl-based extension and parameterization;
- Declaration of signal groups;
- Support of include files.

In order to generate IP test patterns in a modular way and avoid massive changes when a single register address or port range in an IP is changed, we suggest to have 3 separate views:

- register.mpl – contains two types of information: 1. Mapping between register names and their addresses, 2. Field names of each registers. An example of register.mpl is shown in Fig. 3. For instance, register PGSR0 has address 00d, while fields of PGSR0 are presented in register_bits part (0th bit is IDONE, 1st bit is PLDONE, 2nd bit is DCDONE, etc.).
- test.mpl – Sequence of test operations. Example of test.mpl is shown in Fig. 4. As it is seen from the figure, the test patterns are described using high-level operations, like WRITE, READ, DELAY, etc.

```
## array set ::registers {
##                       PGSR0           00d\
## ..........................................
##                       }


## array set ::register_bits {PGSR0 {IDONE 0 PLDONE 1
##                           DCDONE 2 ZCDONE 3 DIDONE 4} \
## ..........................................
##                          }
```

Fig. 3. Example of register.mpl

```
## if {$pattern_name eq {TP2_LB_AC_W_0}} {

## SEL_SEGMENT parallel_if
## Label  "BISTRR : Reset"
## WRITE BISTRR    3
## ..................
## WRITE BISTRR    FC0A001
## DELAY #1000
## READ  BISTGSR
## Label  "READ_CAPTURE BISTGSR: BDONE=1"
## READ_CAPTURE BDONE=1
## }
```

Fig. 4. Example of test.mpl

- setup.mpl – Contains definitions of high-level operators (READ, WRITE, etc.) used in test.mpl. Implementation of those operations depends on test protocol (e.g., APB protocol). Part of WRITE procedure implementation used in test.mpl is shown in Fig. 5.

```
## proc WRITE {addr data} {
## global ::registers
## global ::delay
## global ::pattern_space_length
shift_wr = "1";
.....................
shift_data_in = "$data";
shift_psel_rqvld = "1";
.....................
capture_data_out = "xxxxxxxxxxxxx";
.....................
capture_cfg_qvld = "x";
shift_cfg_qvld = "x";
## }
```

Fig. 5. Example of setup.mpl

4. Automated flow for generating IP level information

Fig. 6 shows the diagram of the automated flow which ensures the following steps:

- Nowadays IPs may have a huge set of ports and usually it becomes time-consuming task to describe all these ports manually. Since this information is available in IP (e.g. in IP top Verilog view), we have developed a conversion utility which translates the port information from IP view (can be in Verilog/SystemVerilog or VHDL format) into MASIS format.
- register.mpl – Similar to ports, nowadays IPs may have a huge set of registers and usually IP vendors provide that information using IP-XACT view [8]. IP-XACT is in XML format which sometimes provides difficulties to integrate that information in a SoC test environment where TCL/Python/Perl or other scripting languages are used. Therefore, we have developed a conversion utility which translates IP-XACT information into register.mpl.
- test.mpl - Since test pattern of nowadays IPs may have more than 1000 operations per test pattern, generating test.mpl manually is a time-consuming process. Usually IP test benches provide information on sequences for each test pattern (e.g., Verilog, STIL, VCD, simulation log file, etc.). We have developed a conversion utility that converts test pattern information into MPL from IP test bench outputs. One of the preferred views for conversion is simulation log file which usually contains enough high-level information about sequences of a given test pattern.
- Since setup.mpl is test protocol specific, it is not automated. But the development of setup.mpl is not a time-consuming task as well as it is one-time effort for each test protocol.
- Once all the necessary views are ready, Generator will generate a one common MASIS view containing IP level information (ports, chains, test patterns, etc.).

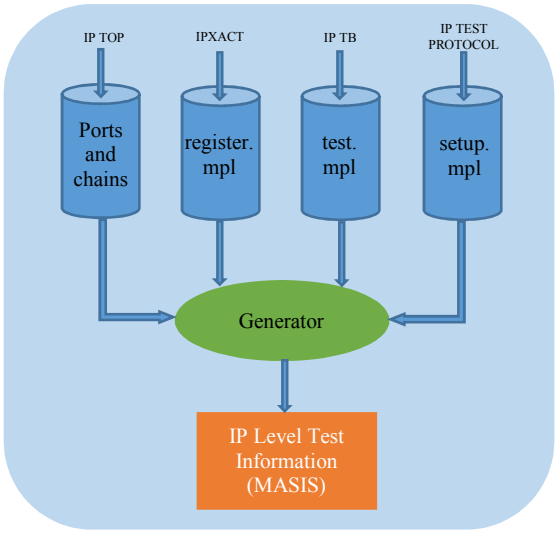

Fig. 6. Automated flow for generating IP level information

5. Experimental results

In order to show the effectiveness of the proposed method, experiments on LPDDR4 IP have been performed. This IP has following tests:

1. TEST_DLL_LINEARITY

- Tries to ensure that the delays generated out of LCDL are linearly increasing or decreasing.
- Contains a sequence of 2200 test operations.

2. TEST_PLL
   - Checks if PLL initialization is done properly.
   - Contains a sequence of 40 test operations.
3. TEST_PULL_DOWN_DQ
   - Brings up PHY and runs continuous calibration at the background, sets the device in Flyover mode, sets the desired effective resistance and sweeps Pull Down impedance with corresponding values.
   - Contains a sequence of 10000 test operations.
4. TEST_PULL_UP_DQ
   - Brings up PHY and runs continuous calibration at the background, sets the device in Flyover mode, sets the desired effective resistance and sweeps Pull Up impedance with corresponding values.
   - Contains a sequence of 10000 test operations.
5. TEST_VIH_VIL_CHAR
   - Tests BP_ALERT_N Receiver.
   - Contains a sequence of 400 test operations.
6. TEST_VREF_DAC0
   - Tests DQ Receiver by enabling DFE0.
   - Contains a sequence of 300 test operations.
7. TEST_VREF_RxDAC1
   - Tests DQ Receiver by enabling DFE1.
   - Contains a sequence of 300 test operations.

So, overall there are 23240 test operations for testing LPDDR4. In order to manually create MPL view for these tests it would take approximately one week. While the proposed automation allows to generate the same MPL views much faster:

- Port and chain information – 1 second;
- register.mpl – 1 second;
- test.mpl – 10 seconds;
- Generate MASIS – 10 seconds.

Since setup.mpl is test protocol specific, it is not automated. From other side, development of setup.mpl is one-time effort for each test protocol and requires about 1-hour effort. So, using the proposed automation it was possible to reduce the development of LPDDR4 IP level test information from 1 week to 1 hour.

6. Conclusion

In this paper, first short introduction to test solutions for SoCs is presented. Then an architecture for hierarchical test system and challenges related to system input data preparation are described. Finally, an automated flow for input data generation is proposed which is based on converting the necessary information from the IP existing views. The presented experimental results demonstrate the efficiency of the proposed approach.

7. References

[1] H. G. Kerkhoff, J. Wan, “Dependable Digitally-Assisted Mixed-Signal IPs Based on Integrated Self-Test & Self-Calibration”, IEEE International Mixed-Signals, Sensors and Systems Test Workshop (IMS3TW), 2010, pp. 1-6.

[2] G. Li, J. Qian, Q. Yang, “Hybrid Hierarchical and Modular Tests for SoC Designs”, IEEE North Atlantic Test Workshop (NATW), 2015, pp. 11-16.

[3] Y. Zorian, S. Shoukourian, “Test Solutions for Nanoscale Systems-on-Chip: Algorithms, Methods and Test Infrastructure”, International Conference on Computer Science and Information Technologies (CSIT), 2013, pp. 1-3.

[4] B. Keller et al., “Efficient Testing of Hierarchical Core-Based SOCs”, IEEE International Test Conference (ITC), 2014, pp. 1-10.

[5] 1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture.

[6] 1687-2014 - IEEE Standard for Access and Control of Instrumentation Embedded within a Semiconductor Device.

[7] F. DaSilva, Y. Zorian, L. Whetsel, K. Arabi, R. Kapur, “Overview of the IEEE P1500 Standard”, IEEE International Test Conference (ITC), 2003, pp. 988-997.

[8] 1685-2014 - IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tool Flows.