# For ultra-large-scale SoC (System on Chip)…

# DFT design is becoming more and more complex along with the scale increasing of SoC.

# To make sure high yield and smoothly post silicon debug, DFT logic functionality should be fully verified like the logic function of the SoC.

# **Introduction:** In DFT (Design For Testability) domain, the test patterns running on an ATE (Automatic Test Equipment) can be categorized into two types: scan related and non-scan related. The former can be generated using ATPG (Automatic Test Pattern Generation) tools, while the latter cannot. These non-scan DFT function tests, like other function tests, are normally created by design verification engineers using languages such as System Verilog or C++. However, ATEs need test patterns described by STIL (Standard Test Interface Language) or other test languages.

# To fill the gap, there is usually a dedicated team to transfer function simulation to ATE test environment, or alternatively in-house automation flows are developed to enforce complex rules on test writing and register specification documentation, which are specific for a given environment and difficult to migrate.

# This paper provides a universal and more efficient solution by introducing a UVM (Universal Verification Methodology) based DFT verification environment that naturally generates test patterns in STIL format during simulation and can be plugged into any UVM based environment. This method applies to other formats that ATEs need.

# For ultra-large-scale SoC (System on Chip), IEEE 1149 protocol alone cannot satisfy DFT design requirements, so IEEE 1687 and 1500 protocols are usually adopted to enable modular and hierarchical DFT test access, leading to challenges when writing test sequences at register abstract level, as different protocol TDRs (Test Data Register) are hierarchically located in a network connected via IEEE 1687. To access a TDR one or more levels 1687 SIBs (Segment Insertion Bit) have to be set and the length of DR (Data Register) chain varies with SIB values. The author also comes up with a general way to model complex DFT test access network.

# This paper is intended to proceed as follows:

# **Concept:**

# **How to Design a DFT Test Bench Can Generate Test Patterns During Simulation:**

# 

# The STIL test pattern describes test stimulus using vectors which specify pad toggle and measurement information (called STIL information hereinafter) in a time period.

# The idea of getting a STIL test pattern comes out of the fact that a drive knows the exact time to drive or sample a certain pad.

# A driver is the best supplier of STIL information since it knows the exact time drive or sample a certain pad.

# If we can collect all the pad drive and sample information during simulation then we probably can resolve the problem.

# The STIL generator collects STIL information from drivers and writes it out according to time stamp property in stil\_info\_transaction.

# The STIL generator is a uvm\_subscriber extended component which collects STIL information from drivers through analysis ports and writes them out according to time stamp. To get STIL information, it’s necessary to let drivers take charge of all pad toggle, no any direct pad connection in test bench. Pads are divided into five categories which correspond to jtag\_driver, clock\_diver, reset\_driver, scan\_chain\_driver and pad\_driver. Since clock pads and reset pads connection are usually simple, they get toggle information from configurations directly others are from sequecne\_items.

# Pads of a SoC can be divided as following types in DFT functional simulation.

# IEEE 1149.1 compliance on-chip TAP (Test Access Port). Hereinafter it’s simply called JTAG (Joint Test Action Group) interface, which is the most significant interface for DFT design.

# Scan related pads. Hereinafter it’s simply called scan interface, which is a group of pads used as scan control signals and scan channels when a chip is in scan test mode.

# Clock pads. It’s the clocks need toggling in DFT functional simulation. Please see BOZO for more description.

# Reset pads. It’s the asynchronous resets of a chip.

# Other pads. Except for type 1-4 mentioned above, the rest pads are categorized as one type. These pads may need have initial value or be toggled once in a while in simulation.

# Each type of pads has a specific driver extends from uvm\_driver to interact with them. In Figure1, jtag\_driver drives JTAG interface, scan\_driver drives scan interface, clock\_driver drivers clock pads, reset\_driver drivers reset pads and pad\_drivers drives other pads.

# Firstly, stil\_info\_transaction sequence item is defined as Figure 2.

# class stil\_info\_transaction extends uvm\_sequence\_item;

# string stil\_info;

# string comment\_info;

# ……

# endcalss

# stil\_info is the pad drive or sample information at a certain time. For example,

# Secondly, add an analysis port, which is an object of uvm\_analysis\_port class specialized with stil\_info\_transaction type, in each driver, except for scan\_driver. Scan test pattern generation, which are normally generated by ATPG tools, is not the topic of this paper.

# Thirdly, a component named stil\_generator, which extends from uvm\_subscriber class specialized with stil\_info\_transaction type, is created. Add correspondent analysis export for each driver in stil\_generator. Since uvm\_subscriber class has only one built-in analysis export, the `uvm\_analysis\_imp\_decl macro need to be used to declare analysis imp export and its associated write() method for the rest analysis export[2].

# Lastly, connect analysis port and analysis export pair as show in Figure 1 in orange lines.

# **Abstract**

# The DFT design is becoming more and more complex accompany with the scale increasing of SoC. How to verify DFT logic completely in simulation and supply test patterns for ATEs with highly coverage is important for post-silicon debug and yield increase.

# While verification methodology is evolving and innovating and has entered UVM era, DFT verification need to keep in pace to leverage the advantage of UVM, therefore to increase test reusability, extendability and function coverage and etc.

# This paper presents a general UVM based DFT verification environment, which is usable from modular DFT verification to SoC DFT verification, and it can generate STIL test patterns for ATE test during SoC simulation.

# Also, this paper presents a method to model hirerarchy abstract DFT TDR

# **Introduction**

# … review..

# why

# This paper is intended to proceed as follows: Section 1.2 will be … Section1.3 …

# Concept

figures

advantage

Implementation

step

code

experience

results, discussion and conclusion

Discussion

judge

effects

insights

sharing

conclusion