AUTOMATED CONNECTIVITY TEST CREATION FOR SYSTEM-IN-PACKAGE ANALOG MIXED-SIGNAL VERIFICATION

Samantha Pandez **Christopher Geen**
Analog Devices Inc.

2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 27-MARCH 2, 2023

INTRODUCTION

Verification is one of today's most important product development challenges. The latest Wilson Research Group Functional Verification Study [1], shows the percentage of ASIC development spent on verification averages around 50-60%, higher than prior year. The same study found verification engineers spend most of their time debugging, creating tests, and running simulations. Fig. 1 from the same study shows these tasks, combined, consume 68% of total verification time. New verification methodologies promising to reduce errors during code development.

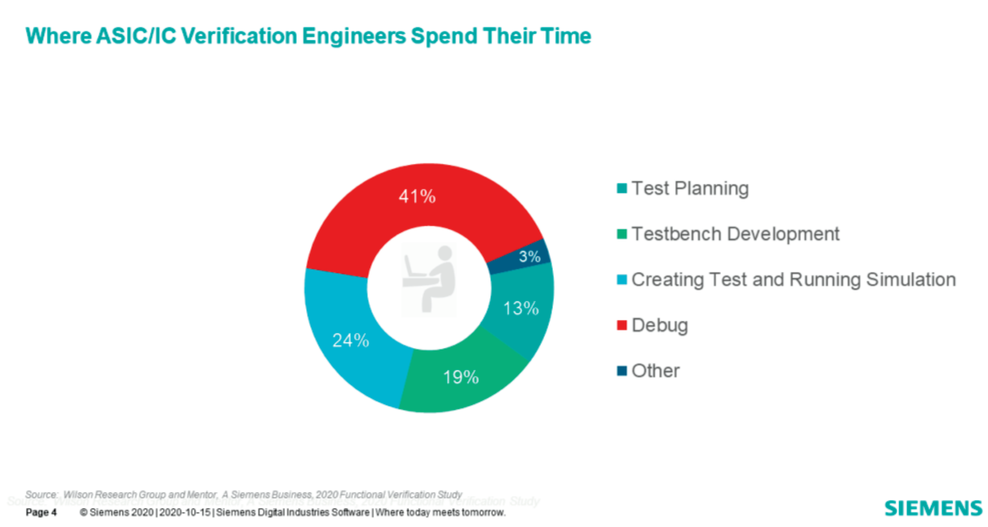

Figure 1. Where ASIC/IC Verification Engineers Spend their time (Siemens EDA and Wilsons Research Group, 2022)

Connectivity checking verifies the intended connectivity from digital pins to their analog circuit destinations, ensuring correct voltage levels and logic states. These tests catch errors such as switched pin positions, incorrect node names, unmatched bus bit positions, and improper reset states. This task is challenging, especially with large complex designs containing multiple levels of hierarchy, gating conditions, and register logic. Manual test creation is error prone, as large designs contain thousands of connections with varying stimulus conditions, all of which are important to ensure that the design will work as intended.

TRADITIONAL MANUAL CONNECTIVITY CHECKING

Modern SiP designs contain complicated subsystems with complex IP blocks and numerous configurable modules, making manual connectivity test development challenging. Writing checkers for every digital logic signal and manually confirming each signal arrives at the proper destination with the intended voltage level is daunting since, checkers must be hand crafted for every relevant connection. This is possible with small designs but becomes untenable with complex products. Complexity invites code errors which impact schedule and productivity. **Our proposed automated approach reduces code errors resulting in increased productivity and efficiency.**

AUTOMATED CONNECTIVITY TEST CREATION

Connectivity tests, including voltage checkers, can be automatically generated using a standardized format consisting of a human readable spreadsheet which is operated on by a Python script producing System Verilog code, as shown on the process in **Figure 2**.

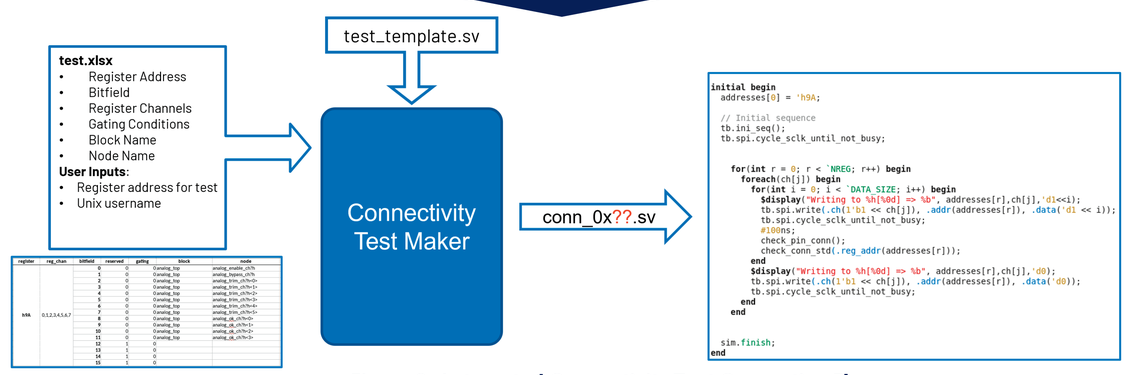

Figure 2. Automated Connectivity Test Generation Flow

```
if (ch[j] == 0 || ch[j] == 2 || ch[j] == 4 || ch[j] == 6) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_trim_chxh<1>"), .exp(reg_tmp[3]), .vtol(vtol));
end else if  (ch[j] == 1 || ch[j] == 3 || ch[j] == 5 || ch[j] == 7) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_trim_chyh<1>"), .exp(reg_tmp[3]), .vtol(vtol));
end

if (ch[j] == 0 || ch[j] == 2 || ch[j] == 4 || ch[j] == 6) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_trim_chxh<0>"), .exp(reg_tmp[2]), .vtol(vtol));
end else if  (ch[j] == 1 || ch[j] == 3 || ch[j] == 5 || ch[j] == 7) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_trim_chyh<0>"), .exp(reg_tmp[2]), .vtol(vtol));
end

if (ch[j] == 0 || ch[j] == 2 || ch[j] == 4 || ch[j] == 6) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_bypass_chxh"), .exp(reg_tmp[1]), .vtol(vtol));
end else if  (ch[j] == 1 || ch[j] == 3 || ch[j] == 5 || ch[j] == 7) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_bypass_chyh"), .exp(reg_tmp[1]), .vtol(vtol));
end

if (ch[j] == 0 || ch[j] == 2 || ch[j] == 4 || ch[j] == 6) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_enable_chxh"), .exp(reg_tmp[0]), .vtol(vtol));
end else if  (ch[j] == 1 || ch[j] == 3 || ch[j] == 5 || ch[j] == 7) begin
»       check_vlogic(.ch(ch_str), .block("analog_top"), .node("analog_enable_chyh"), .exp(reg_tmp[0]), .vtol(vtol));
end
```

Figure 3. Sample pin checkers generated by the script

- Ensures digital pins are correctly wired to their respective analog blocks guaranteeing no pins are mis-wired.
- Verifies the intended voltage signal levels arrive at the analog block.
- Addresses the complexity of circuit hierarchy by automatically constructings the path.
- This method was successfully implemented on the latest System-in-Package Pin Driver ATE Project at Analog Devices, Inc. where it generated checkers for >2000 interface pins.

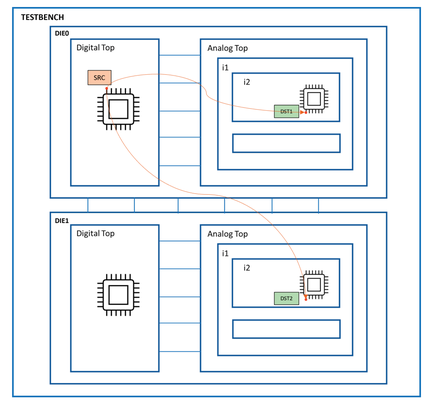

Figure 4. Digital source to its analog block node destination going through several layers of hierarchy for separate die

SUMMARY OF RESULTS

- Produced an alternative to manual code generation whereby a python script auto-generates verification code reducing development time and cost.
- Created connectivity tests with voltage checkers in a span of minutes as opposed to the days it would require when using the typical manual coding approach.
- Completed almost 600 checks during a complex ADI pin electronics project using over 2000 digital pin connections.
- Productivity gains allowed the verification engineer to spend more time on other mission critical tests.

ADVANTAGES AND BEST PRACTICES

Hand crafting hundreds of connections and checkers for read/write transactions takes an extremely long time and is error prone. Our new methodology's quickly produces a standardized test structure which can accurately parse a complex analog hierarchy. Reducing this effort from weeks to minutes. This methodology is flexible accommodating hierarchical changes since re-running the script will generate updated checkers. Many of our projects utilize same register definitions or analog block names, allowing for reuse of a majority of the spreadsheet between projects. Since the hierarchy information is extracted from the netlist, the methodology can be utilized in either a top-down or bottom-up approach. It can also cover blocks which uses SystemVerilog models if it is included on the netlist. The abstraction level changes based on the test requirements.

REFERENCES

[1] Wilson Research Group and Siemens EDA, "2022 Functional Verification Study", November 2022

ACKNOWLEDGMENT

The authors would like to express their gratitude to ADI ATE Design Verification Team - Carl Heerbrugg Ramirez, Roberto Suarez-Valle and Anton Mark Sumollo, for their valuable feedback during the development of this work. The authors would also like to acknowledge the ADI ATE Design Team, and ADGT IMM Leadership for the support, resources, and suggestions on the pursuant of this work.