ATLANTIS PRESS

Advances in Intelligent Systems Research (AISR), volume 145

2017 International Conference on Electronic Industry and Automation (EIA 2017)

An Approach for Checking RTL Design Signal Sources and Destinations

Jianguo SONG
College of Electronic Information and Control Engineering
Beijing University of Technology
Beijing, China
e-mail: sjglcy@126.com

Aojie CHEN
College of Electronic Information and Control Engineering
Beijing University of Technology
Beijing, China
e-mail: chenaojie2015@163.com

Tao GUO
WinnerMicro
Yindu Mansion 18th floor
Beijing, China
e-mail: guotao@winnermicro.com

***Abstract*—Formal check provides a method for detecting problems of SoC design in early time of verification. Looking at SoC design issues, most of design integration errors come from modules connectivity. Combining above two points, this paper descripts an approach check signal connectivity with formal check method. The approach parses Verilog HDL RTL code through a program, which writes by script language Perl, to analysis signals connectivity. It's helpful for both SoC designer and verification engineer.**

Keywords-System-on-Chip; Verilog HDL; RTL; connectivity; formal check; signal connection; Perl

I. INTRODUCTION

Following with increasing of chip scale and design complexity, digital chip integration level advance rapidly. Millions of gate-level System-on-Chip (SoC) design is universal [4]. In order to reduce functional and logical error probability and guarantee accuracy of chip design, improving the efficiency of verification has been come out an issue in this field [5]. The problem of design verification in electronic circuits is solved by means of various Electronic Design Automation (EDA) tools, which could assist verification engineers to going on dynamic simulation and formal check. Compare with dynamic simulation, formal check does not need waveform simulation, this means that the state space could be traverse completely through mathematical method, and more time could be saved. In addition, the issues beyond functional design could be found at early stage of RTL, thus the turnaround time will be cut down [1]. Formal check methods include: syntax check, linting check, cross-clock domain check and formal verification [9]. The methodology we describe in this paper is part of linting check. Linting check, compare with syntax check, pay more attention to design feasibility, it's helpful to enhance code consistency, efficiency and readability, it as well helps to improve design coding style.

SoC projects are usually based on reuse and modification of already existing systems. Re-use past designs is necessary in order to shorten the design cycle by re-using previous work or component libraries. But most of errors during the integration of new devices on the system are due to incorrect connections, cause by inevitable artificial reason [1]. Connection includes module internal link type and pin to pin type between modules, which usually define in top-level modules. Signal is a dataflow oriented pulse, therefore a connection could be described by its source and destination nodes.

In this paper we combining the above two aspects, check signal connection problems with linting check method through a program write by script language. Script always aids verification to solve repetitive situation validly. Without using verification environment, whether the designers or the verification staffs could check design errors distinctly almost at the same time before waveform simulation. Since linting check does not depend on the accuracy of functional description to achievement, it's not necessary to involve System Verilog Assertions (SVA) in code. By this way, the benefits of guarantee design quality it brings are obviously.

II. RELATED WORK

SoC interface connectivity is studied in [1,2]. Using formal check is discussed in [1,3], the advantage of formal check is obvious compared to conventional simulation techniques. The characteristics of RTL code is concluded in [3,4]. In [1] Re-use techniques is discussed, which is necessary in order to shorten the design cycle. At the time [2] summaries two simple generation method for checking connectivity of interface, the one is using a text editor by providing the formal to actual connections, the other is using script or program traverses the set of design modules, attempts to make connections based on specified rules. The method of checking Intellectual Property (IP) blocks is studied in [4,5].

The paper used a mature script language, Perl, which play a nice role in dealing with content matching, to parse Verilog code. In process of achieving the approach, seek Perl function in [10], and confirm Verilog syntax in [6,7]. Besides, formal check is always assisted by dedicated tools, like

synopsys's SpyGlass. SpyGlass is a powerful software recognized by IC industry, it collects a lot functions, such as RTL analysis, CDC check, DFT testability analysis and so on. Compare linting check field with SpyGlass [9], our methodology is more powerful on signal connection, the source and the destination nodes could be marked out clearly.

III. PARSING VERILOG SOURCE FILES

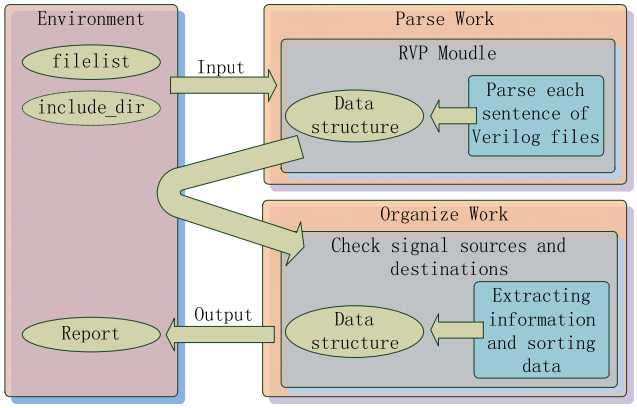

Figure 1. The structure of the program

This section descripts implementation method of the program. Fig. 1 shows the total structure of the program. Environment provides input files, parse platform receives them through relevant interface. After parsing Verilog files, a huge complex data structure will be generated. Organize platform will extract the information of the program needs, and will deal with output format to complete the final report, which is feedback to environment for the user.

Rough Verilog Parser (RVP) is a useful Perl module, a lot of work has been completed in terms of parsing Verilog code. RVP module analysis Verilog code by matching key words on the Verilog syntax, accepts IEEE Verilog standard 1364-1995 and 1364-2001 [6,7]. The function of extracting the relevant information of signal connection, especially signal sources and destinations, have been modified to adapt to the approach, thus to form relative rules. Some examples are shown in follow:

(1) always @(posedge clk or negedge rst_n)

(2) always @(AN or BN or CN)

(3) if (AN[31:29,27,25:0] == BN[0:29])

(4) case (AN[31:29,27,25:0])

(5) wire [15:0] AN = (BN == 32'b0) : 16'b0 ? 16'hfffff, CN, DN = {EN[7:0], FN[15:8]};

(6) assign {AN[7:0], BN[3:0], CN[3:0]} = DN[31, 29, 27:20, 5:0];

(7) AN <= {(BN[7:0] & CN[7:0]), 16'b0};

(8) module_x #(4'b1001) u_module (

.AN ({AN[15, 13, 11, 9], BN[4:0]}),

.BN (BN),

.CN ());

(1) (2) are based on event control rule. On (1), 'clk' and 'rst_n' are edge-triggered signals in event control, therefore this two signals have the destination. Compare with (1), (2) is combinational logic, the signals in brackets are only trigger condition, not sure if there's any assignment behavior below, so there is no destination for signals in this situation.

(3) (4) are based on conditional expression rule. All signals in conditional brackets have the destination. It's important to pay attention that only the involved bits have its destination.

(5) (6) (7) are based on assignment rule. Signals before assignment symbol have the source, besides signals after assignment symbol have the destination.

(8) is based on instantiation rule, which is a special phenomenon in HDL. Bottom module interface signals and signals of connected with it, their source and destination depends on bottom module interface type.

Choosing (6) as an example, the process of running (6) depicted in Fig. 2. The block diagram is part of program section, the arrow show the action performance. Firstly, assignment action should be occurred in module, so 'MODULE' is considered as the beginning of the rule. If the key word 'assign' is matched, program will run into 'ASSIGN' section. Secondly, if signal is matched, after record definite bits, the signal name and other information will be updated into the data structure. Because signals are found before assignment symbol, they have their source. When the symbol '=' is matched, 'ASSIGN_AFTER_EQUAL' section will be traversed. Finally, as same as the process of 'ASSIGN' section, the data structure would be updated, due to the bits of signals destination has been changed.

From the above information, each rule has its own action performance. Going through with all rules, signal source and direction information is obtained from the code.

By the way of all input Verilog files for parsing sentence by sentence, huge system level data structure will be generated. In addition to recording the source and destination of signal each bit, the content of files, module's tasks, functions, parameters, instances etc. information are noted soundly in the data structure. Moreover, the RVP module equipped with a variety of convenient access for checking data from the structure.

Verilog uses the thinking of "top-down". The complex function is always divided into low level modules, and then connections between modules are achieved by instances. On this basis, modules, instance by nothing, are the most top of this system. Starting traverse module signal from the most top modules, through down each level base on the instance data. When bottom level modules were traversed, all modules have been checked. The signal source and destination report will be created immediately after traversing.

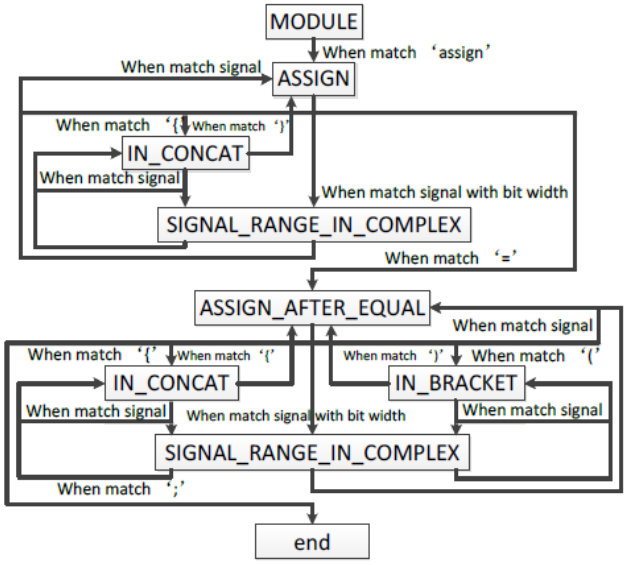

Figure 2. The process of running example (6)

IV. CASE STUDY

The approach presented in this paper for extraction of signal source and destination got applied on a SoC example from a company's RTL code. The company, Winner Micro, is committed to wireless communication chip dedicated Internet of things field development and application. And the code is a RF chip and integrates many kinds of interface protocols in a SoC with a comprehensive documentation in Verilog 1364-2001 format. Information about the SoC design is presented in Table. 1. Fig. 3 descripts a part of input file list.

TABLE I. SOC DESIGN INFORMATION OF THE CASE

Metric	Value
Number of files	314
Number of modules	326
Number of defines	391
Number of instances	928
Number of signals in instances	31949

The output report is divided into two parts. First, listing all top modules in input files, because maybe some redundant code are mixed in them, it could influence the program for checking signals. If the file of module is unnecessary, the path of the file should be deleted from file list.

Second shows in Fig. 4, the connection problems detail of each signal source and destination would be listed, it is content of the dominant. Same as above, module's name, instance path and file path are displayed, and then the line number of signal definition, signal type, signal name, debatable bits and its driver (source) and cause (destination). The result of signal connectivity paths could be shown as well.

While Fig. 5 shows the sum of signal problems in this SoC example, general situation could be understand from it.

```
/simulation/chenaojie/paper/rtl/spi_slave_ctrl.v
/simulation/chenaojie/paper/rtl/clkg_rst_ctr_venus.v
/simulation/chenaojie/paper/rtl/tx_edcf.v
/simulation/chenaojie/paper/rtl/i2s_apb_if.v
/simulation/chenaojie/paper/rtl/dsram2_mmu.v
/simulation/chenaojie/paper/rtl/outputarb_b2.v
/simulation/chenaojie/paper/rtl/isram.v
/simulation/chenaojie/paper/rtl/spi_tx.v
/simulation/chenaojie/paper/rtl/sc_uart_regfile.v
/simulation/chenaojie/paper/rtl/WdogFrc.v
/simulation/chenaojie/paper/rtl/dur_ctrl.v
/simulation/chenaojie/paper/rtl/gpaes_inv_cipher_top.v
/simulation/chenaojie/paper/rtl/ahb_matrix_b1.v
/simulation/chenaojie/paper/rtl/dma_ahb_master.v
/simulation/chenaojie/paper/rtl/IncrOverride.v
/simulation/chenaojie/paper/rtl/bus_cov.v
/simulation/chenaojie/paper/rtl/gpsec_sbox_init.v
/simulation/chenaojie/paper/rtl/Sync1.v
/simulation/chenaojie/paper/rtl/qset.v
/simulation/chenaojie/paper/rtl/mac_mem.v
/simulation/chenaojie/paper/rtl/rx_fifo32_8.v
```

Figure 3. A pert of input file list

```
module   : Ahb2Apb
instance : vn_top_wrapper.u_vn_core_wrapper.u_cpu_domain_wrapper.u_apb_top.u_ahb2apb
filepath : /simulation/chenaojie/paper/rtl/Ahb2Apb.v

224      wire           HaddrMux
   [24]  Driver   has   Cause    X
   [25]  Driver   has   Cause    X
   [26]  Driver   has   Cause    X
   [27]  Driver   has   Cause    X

189      wire           SCANENABLE
   [0]   Driver    X    Cause    X

190      wire           SCANINHCLK
   [0]   Driver    X    Cause    X

195      wire           SCANOUTHCLK
   [0]   Driver    X    Cause    X


module   : i2s_rx
instance : vn_top_wrapper.u_vn_core_wrapper.u_cpu_domain_wrapper.u_apb_top.u_i2s.u_i2s_rx
filepath : /simulation/chenaojie/paper/rtl/i2s_rx.v

369      wire           rx_clk_pre
   [0]   Driver    X    Cause    X

138      reg            rx_fifo_empty_f
   [0]   Driver   has   Cause    X
```

Figure 4. Detail of connection problems

```
#################################################################
Attention : Driver Miss         : 745
            Cause  Miss         : 4062
            Miss both of above  : 681
            Total  Error        : 5488
#################################################################
```

Figure 5. Sum of signal problems in case

V. CONCLUSION

A large percentage of SoC integration errors comes from modules or IPs connectivity issues. Verification of signals connectivity is one of the problem domains which Formal Verification is very good at solving compared to the conventional simulation techniques. The earlier RTL design errors found, the less turnaround time wasted. This paper presents an approach for judging signal connectivity. The approach is based on Verilog language rules, extracts signal information in the code, and tracks signal sources and destinations to find connectivity problems. The approach obtains a good result on a RTL code, helps designers rule out the hidden trouble of connective problems.

Interface connections is one of connection type, if the input interface is empty, the logic may be influenced by unknown value, like 'X'. The wire which link to internal signal of module is the other connection type, the wire maybe has no source, no destination or neither of them. Although this signal is not wrong, and will be ignored by the simulator, it not only likes to develop into tricky trouble, but also interfere with the verification engineer. In addition, it has effect on enhancing code style too.

REFERENCES

[1] Haytham Saafan, M. Watheq El-Kharashi, Ashraf Salem, "SoC connectivity specification extraction using incomplete RTL design: an approach for Formal connectivity Verification", 11th International Design & Test Symposium (IDT), 2016, Page(s): 110 - 114, DOI: 10.1109/IDT.2016.7843024.

[2] Cristian-Győző Haba, Derek Pappas, "CSLC: The infrastructure compiler for SoC design", International Conference on Development and Application Systems (DAS), 2014, Page(s): 149 - 154, DOI: 10.1109/DAAS.2014.6842445.

[3] Carlos Ivan Castro Marquez, Marius Strum, Wang Jiang Chau, "A unified sequential equivalence checking approach to verify high-level functionality and protocol specification implementations in RTL designs", 15th Latin American Test Workshop – LATW, 2014, Page(s): 1 - 6, DOI: 10.1109/LATW.2014.6841905.

[4] Bhanu Singh, Arunprasath Shankar, Francis Wolff, Christos Papachristou, Daniel Weyer, Steve Clay, "Cross-correlation of specification and RTL for soft IP analysis", Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014, Page(s): 1 - 6, DOI: 10.7873/DATE.2014.303.

[5] Tomas Grimm, Djones Lettnin, Michael Hübner, "Automatic generation of RTL connectivity checkers from SystemC TLM and IP-XACT descriptions", IEEE Nordic Circuits and Systems Conference (NORCAS), 2016, Page(s): 1 - 6, DOI: 10.1109/NORCHIP.2016.7792922

[6] IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language, 1364-1995.

[7] IEEE Standard Verilog Hardware Description Language, 1364-2001.

[8] rvp - Rough Verilog Parser Perl Module [Online]. Available: http://www.burbleland.com/v2html/rvp/rvp.pod.html.

[9] SpyGlass® Predictive Analyzer User Guide, Atrenta, Inc. Available: http://www.atrenta.com.

[10] perldoc online http://perldoc.perl.org/.