Towards Debug Automation for Timing Bugs at RTL

Mehdi Dehbashi*
*Institute of Computer Science, University of Bremen
28359 Bremen, Germany
Email: dehbashi@informatik.uni-bremen.de

Görschwin Fey*†
†Institute of Space Systems, German Aerospace Center
28359 Bremen, Germany
Email: goerschwin.fey@dlr.de

***Abstract*—One major concern in the design of *Very-Large-Scale Integrated* (VLSI) circuits is debugging as design size and complexity increase. Automation of the debugging process helps to decrease the development cycle of VLSI circuits and consequently to achieve a higher productivity. This paper presents an approach to automatically debug timing bugs at the design step. The approach utilizes *Boolean Satisfiability* (SAT) in order to model design timing bugs at the pre-silicon stage. The experimental results show diagnosis accuracy and efficiency of the approach.**

***Keywords*—debug automation, timing bug, diagnosis accuracy**

I. Introduction

Due to the increasing design size and complexity of VLSI circuits, the cost of VLSI systems verification and debugging has significantly increased. Verification tools check the correctness of a design against its specification. Upon detection of a design error, the error is returned as a counterexample. Having a counterexample, the debug process starts localizing and rectifying the bug. But this process is often a manual task which needs a large effort. Thus, automated approaches to design debugging are necessary to decrease the development cycle of VLSI products.

Design bugs at RTL are classified into three major classes: *logic bugs*, *algorithmic bugs* and *timing bugs* [1]. There is a range of approaches to automate the debugging process for logic bugs [2] [3] [4]. Algorithmic bugs can have a severe impact on the correctness of a design and they usually require multiple major modifications to be fixed. Timing bugs are related to the correctness of the timing behavior in a design. For most of the timing bugs, a signal requires to be latched a cycle earlier or a cycle later in order to keep the correct timing behavior of signals in the design [1]. The most common fix for this class of design bugs is the manual addition or removal of flipflops to satisfy the correct timing behavior of the circuit. These bug models are called *missing flipflop* and *extra flipflop*.

In the pre-silicon stage, a design is verified against its specification by verification tools. A specification describes the correct timing behavior of a design. The work in [5] presents a formal method to specify the relations between multiple clocks and to model the possible behaviors. Then, a hardware design is verified against the specified clock constraints. An efficient clock modeling approach is presented in [6] to handle clock related challenges uniformly. Clock constraints are automatically generated to avoid unnecessary unrolling and loop-checks in *Bounded Model Checking* (BMC).

This work was supported in part by the European Union (Project DIAMOND, FP7-2009-IST-4-248613) and in part by the German Research Foundation (DFG, grant no. FE 797/6-1).

The work in [2] presents a model based on Boolean satisfiability to automate debugging of logic bugs. A circuit is enhanced with correction logic in order to find the potential fault candidates. The work in [3] uses randomly generated counterexamples for debugging and applies automatic correction based on re-synthesis. An exact debugging approach based on *Quantified Boolean Formulas* (QBF) is proposed in [4]. That creates high quality counterexamples to find fault candidates fixing any erroneous behavior. In [7], a pre-silicon debugging flow is proposed for testbench-based verification environments. The approach uses diagnostic traces to obtain more effective counterexamples and to increase the diagnosis accuracy. All of the mentioned works consider logic bugs in order to automatically localize and to rectify an erroneous behavior at the pre-silicon stage.

In this paper, we present an approach to automate the debugging of timing bugs at RTL at the granularity of clock cycles. First, timing bugs (extra/missing flipflop) are modeled and converted into a Boolean satisfiability formula. Having a counterexample given by a verification tool, our approach automatically extracts potential fault candidates which explain the erroneous behavior of the corresponding counterexample. The erroneous behavior can be fixed by inserting or removing some flipflops in the circuit. The experimental results show effectiveness and diagnosis accuracy of our approach.

The remainder of this paper is organized as follows. Section II presents our debugging approach and explains how to model the correction of timing bugs in order to automatically debug a design using Boolean satisfiability. Section III presents experimental results on benchmark circuits.

II. Approach

At the design step, verification tools check the correctness of an implemented circuit according to the specification. If there is a contradiction between the behavior of the implemented circuit and the specification, this contradiction is returned as a counterexample.

Having an erroneous behavior (counterexample) caused by a timing bug, debugging starts. First the circuit is unrolled as many times as the number of clock cycles constituting the corresponding counterexample. For example, if the length of the counterexample is three clock cycles, the circuit C is unrolled three times: C_0, C_1, and C_2. In this case, the input of a flipflop from clock cycle i is connected to the appropriate gates in clock cycle $i+1$. In the example circuit of Figure 1, there is a missing flipflop bug. The location of the bug is shown by a red circle. To debug the circuit, it is copied three times. The input of flipflop A at cycle i is connected to the appropriate signal at cycle $i+1$. For debugging, we investigate

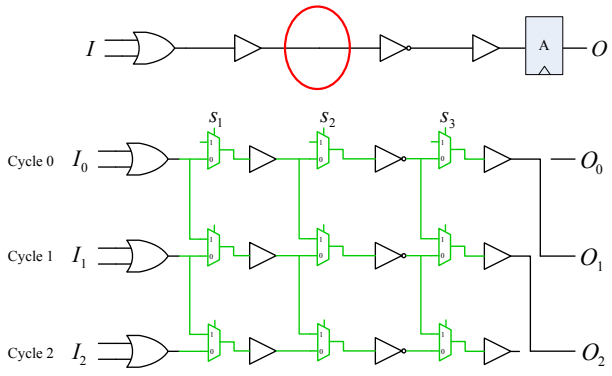

Fig. 1. Debugging Instance for Missing FF Bug

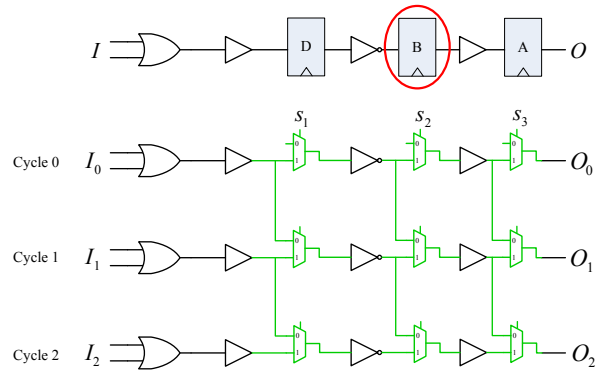

Fig. 2. Debugging Instance for Extra FF Bug

at which point of the circuit a flipflop is missing. Therefore, we need correction logic at each point of the circuit which is able to model the behavior of a flipflop at the corresponding point. The green part in Figure 1 shows this model. Multiplexers are utilized to model a flipflop behavior at every point of the circuit.

If select line s at point p is active ($s = 1$), a flipflop behavior at point p is activated. Therefore, the input of the activated flipflop at cycle i is connected to the appropriate gates at cycle $i + 1$. If the correction logic is inactive ($s = 0$), the circuit at point p has the behavior of a normal wire.

Having correction logic at every point of the circuit, the inputs and output of the model are constrained according to the inputs and output values of the corresponding counterexample. Then debugging answers the following question by activating the select lines: If a flipflop is added at point p, can the erroneous behavior of the corresponding counterexample be fixed? In this case, a SAT solver is utilized to extract all possible fault candidates.

Figure 2 shows the model for the extra flipflop bug. In the case, correction logic is applied at the location of every flipflop in the circuit. Correction logic for the extra flipflop bug has the reverse behavior in comparison to correction logic for the missing flipflop bug. For this kind of bug, debugging investigates if a flipflop is removed, can the erroneous behavior of the corresponding counterexample be fixed?

TABLE I
RESULTS FOR SINGLE FAULTS

Benchmarks			Missing FF			Extra FF		
Circuit	#Gates	#FF	#FC	Time	Mem	#FC	Time	Mem
b01	49	5	2	607	16	1	301	15
b02	25	4	9	1297	15	3	503	15
b04	707	66	4	2228	23	1	297	16
b05	1054	34	6	981	23	2	403	16
b08	177	21	2	1117	18	2	397	16
b10	211	17	1	1043	18	1	301	16
b11	790	31	1	1184	24	2	421	17
b12	1062	121	8	19487	28	5	696	19
gcd	1012	59	5	892	22	1	314	18
phase.	1672	55	8	2838	35	2	467	21

III. EXPERIMENTAL RESULTS

In this section, we demonstrate the effects of our debugging approach experimentally. The techniques described in this paper are implemented using C++ in the WoLFram environment [8] and are evaluated on sequential circuits of LGsynth93 and ITC-99 benchmark suites. The single faults are randomly injected by removing a flipflop (Missing FF) or by adding a flipflop (Extra FF).

The experiments are carried out on a Dual-Core AMD Opteron(tm) Processor 2220 SE (2.8 GHz, 32 GB main memory) running Linux. MiniSAT is used as underlying SAT solver [9]. Run time is measured in CPU seconds, and the memory consumption is measured in MB.

Having a buggy circuit, the verification process returns an initial counterexample. Then, our debugging approach finds the timing fault candidates. Table I presents the experimental results for single faults. The table shows the characteristics of the benchmarks (columns 1-3), final number of fault candidates ($\#FC$), the required run time ($Time$), and the maximum memory consumption (Mem).

REFERENCES

[1] K. Constantinides, O. Mutlu, and T. M. Austin, "Online design bug detection: RTL analysis, flexible mechanisms, and evaluation," in *International Symposium on Microarchitecture (MICRO)*, 2008, pp. 282–293.

[2] A. Smith, A. Veneris, M. F. Ali, and A. Viglas, "Fault diagnosis and logic debugging using boolean satisfiability," *IEEE Trans. on CAD*, vol. 24, no. 10, pp. 1606–1621, 2005.

[3] K. Chang, I. Markov, and V. Bertacco, "Fixing design errors with counterexamples and resynthesis," in *ASP Design Automation Conf.*, 2007, pp. 944–949.

[4] A. Sülflow, G. Fey, and R. Drechsler, "Using QBF to increase accuracy of SAT-based debugging," in *IEEE International Symposium on Circuits and Systems*, 2010, pp. 641–644.

[5] E. M. Clarke, D. Kroening, and K. Yorav, "Specifying and verifying systems with multiple clocks," in *Int'l Conf. on Comp. Design*, 2003, pp. 48–55.

[6] M. K. Ganai and A. Gupta, "Efficient BMC for multi-clock systems with clocked specifications," in *ASP Design Automation Conf.*, 2007, pp. 310–315.

[7] M. Dehbashi, A. Sülflow, and G. Fey, "Automated design debugging in a testbench-based verification environment," in *EUROMICRO Symp. on Digital System Design*, 2011, pp. 479–486.

[8] A. Sülflow, U. Kühne, G. Fey, D. Große, and R. Drechsler, "WoLFram – a word level framework for formal verification," in *IEEE/IFIP Int'l Symposium on Rapid System Prototyping*, 2009, pp. 11–17.

[9] N. Eén and N. Sörensson, "An extensible SAT solver," in *SAT 2003*, ser. LNCS, vol. 2919, 2004, pp. 502–518.