行政院國家科學委員會專題研究計畫　成果報告

以資料路徑為基礎之超大型積體電路暫存器轉移層次描述的驗證與診斷方法

計畫類別：個別型計畫
計畫編號：NSC93-2215-E-032-002-
執行期間：93 年 08 月 01 日至 94 年 07 月 31 日
執行單位：淡江大學電機工程學系

計畫主持人：饒建奇

報告類型：精簡報告

處理方式：本計畫可公開查詢

中　華　民　國 94 年 12 月 15 日

A Data-Path Based Diagnosis Mechanism for RTL Description of VLSI Circuits

Jiann-Chyi Rau, Yi-Yuan Chang and Chia-Hung Lin
Department of Electrical Engineering, Tamkang University
151, Ying-Chuan Rd. Tamsui, Taipei Hsien 251, Taiwan, R.O.C
{jcrau , yychang, chlin}@ee.tku.edu.tw

Abstract

In this paper, an efficient algorithm to diagnose design errors in RTL description is proposed. The diagnosis algorithm exploits the hierarchy available in RTL designs to locate design errors. Using *data path* to reduce the number of error candidates and ensure that true errors are included in. According to the estimated probability, the most suspected error candidates would be reported first in the display. The advantages of the proposed method are simple and available.

Keywords: Data path, Register-Transfer Level, gate level, Hardware Description Language.

1. Introduction

The speed and complexity of digital circuits has increased rapidly. Designers have responded by designing at higher levels of abstraction. The increasing complexity of VLSI circuit designs, debugging /diagnosis represents an important cost in design development.

Much of the previous work on error diagnosis has primarily been targeted at the gate and lower levels of design. Traditionally, in the problem of design error diagnosis, the implementation is often represented as lower level (gate level) circuits and the specification is defined as higher level (RTL) circuits. However, most design activity presently takes place at the RTL and it is difficult to relate errors at the RTL to errors at a lower level. In fact, a relatively simple error at the RTL may translate into extremely complex errors at a lower level. Hence, it is critical to address the diagnosis problem at the RTL.

In modern design process, most design errors occur in the early stage of describing the functional behavior of a design in HDLs and tracing the code manually often performs design error diagnosis at this stage. However, for modern designs with thousands of lines of HDL code, debugging such circuits manually is a difficult task. Therefore automatic design error diagnosis techniques for HDL designs are proposed [1][2][3]. In [1], Vamsi Boppana et al exploits hierarchy available in RTL design to locate design errors and the information from the simulation of Xlists to capture the effects of design errors within components of RTL designs. Maisaa Khalil et al [2] proposed an approach to point out the exact or likely error location with the assumption that both the set of test cases and the corresponding simulation results are available. However, the number of the error candidates may still be too large for designers to debug.

In this paper, the propose *data path* approach to search for design error diagnosis for HDL coding in compared with [3]. The proposed framework reduce the number of error candidates while ensuring true errors are included in. According to the estimated probability, the most suspected error candidates will be reported first in the display such that the efforts of debugging can be further reduced.

The paper is organized as follows. Section 2 introduces the *data path* and *data path digraph*. Section 3 gives the work overview, describes the reduction of error candidates and estimates the probability of correctness for each potential error candidate. Finally, the experimental results in section 4 and the conclusions in section 5.

2. Data Path

The architecture of data paths treated in this paper is based on a *multiplexed data path architecture* [7], can regard such a *data path* as a concatenation of hardware elements and lines. A hardware element is an operational module (OP), a primary input (PI), a primary output (PO), a register (Reg), a multiplexor (MUX). An operation module is a combinational circuit and includes no register. Values enter into a hardware element through its input ports, and exit through its output port. Each line connects between one input port of a hard ware element and one output port of another. Any number of lines can connect to an output port, but only one line can connect to an input port. The restrictions the data path as follows:

- For any input port, there exists a path from a primary input.
- For any output port, there exists a path to a primary output.
- An operation module has only one or more input ports and only one output port.

Further, using a *data path digraph* to represent structure of a data path [4]. Fig. 1 illustrates a data path and its data path digraph. Be careful of the dotted line represent the MUX's decision.

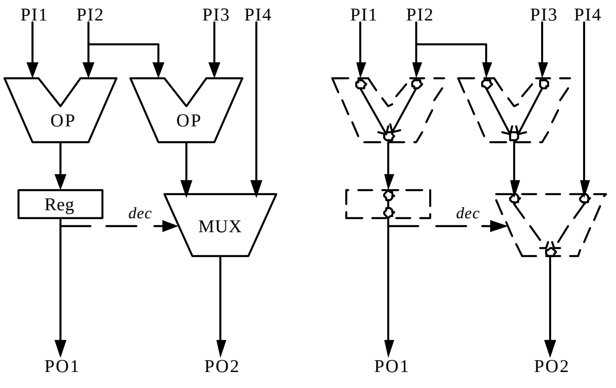


Figure 1 .A data path and its data path digraph

3. Work Overview

User must given something as follows:

- Synchronous digital HDL coding.
- Expected values of all POs.
- Can demonstrate erroneous effects of test pattern.

Give a synchronous digital HDL coding, which is given as the expected values of all POs and registers at all clock cycles, and test patterns that can demonstrate erroneous effects. The approaches use *data path* to check the simulation values of all POs in each clock cycle. If mismatches between the simulation values and the expected values, we take the faulty HDL coding as inputs and output the set of error candidates in an order, which is form the most suspected one to the most innocent one in data path. If not, must continue the simulation and the error checking until run off the test pattern. The data flow is shown Fig. 2.

3.1 Identification Error Space

In this section, we will show find the error space in this thesis. All statements in the error space are potential error sources and may cause design errors. Because the error space is used to help designers identify errors in the design and correct them, the true error sources should be included in the error space. Therefore, the primary concern while finding the error space is to ensure that true error sources are included in it. If we have tried our best but still cannot judge whether the statement is erroneous or not, we prefer to keep this statement in the error space to avoid losing any possible error source. In other words, to make sure that the true error sources are included in the error space is much more important than the size of the error space for an effective error diagnosis.

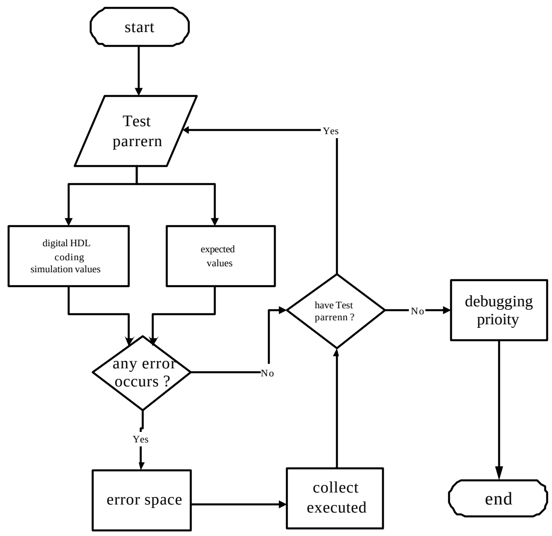


Figure 2. A data flow

Our goal is using simple graph (data path) to minimize the size of error space and ensuring true errors are included in. The reduction of error space can be very helpful because its size directly corresponds to the efforts of debugging. Although the size is not the first concern for an error space, we will still try to make it as small as possible. The smaller error space means less efforts to find the design errors and would be more helpful to designers. There are five steps of our approach as follows:

1. Draw up a whole data path.
2. Search for mismatches PO, and back-trace form PO to PI according to the relationship in the whole data path.
3. Executed statements of error occurring clock cycle, in which an error appears for the first time to search for variation, PI.
4. The correspondence values of PIs will remove in data path.
5. Search out incomplete OP and make up it in data path.

In order to explain the above-mentioned five steps more clearly, this paper use the Verilog code shown in Fig. 3 as an example. In Fig. 3 the code is correct design that designers expect. But for some reasons, the statement S3 is written incorrectly and becomes "*r1 = PI1 & PI2;*" in original coding. Because the simulation values and the expect values are not corresponding so have an error occurs at PO1 at 30ns. The simulation values and the expect values of POs are shown in Fig. 4.

```
module com (PI1,PI2,PI3,PI4,PI5,PO1,PO2,PO3);
output    PO1,PO2,PO3;
input     PI1,PI2,PI3,PI4,PI5;

assign    se11 = PI2 ^ PI5;              //S1
assign    sel2 = PI1 & PI4;              //S2
assign    r1 = PI1 | PI2;                //S3
assign    r2 = PI4 & PI5;                //S4
assign    PO3 = r2 ^ PI5;                //S5
assign    PO2 = (sel2)? PO1 : r2;        //even2,dec2
always @(sel1 or r1 or PI3)              //even1
begin
          if (sel1)                      //dec1
                    PO1 = r1;            //S6
          else
                    PO1 =PI3;            //S7
end
endmodule
```

Figure 3. Verilog HDL coding

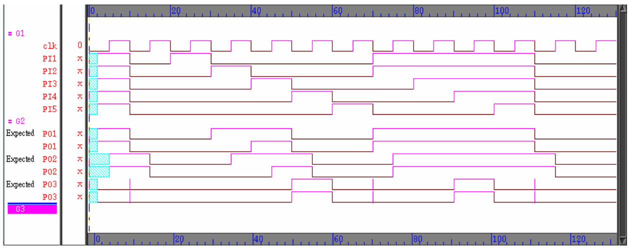


Figure 4. The waveform of Fig.3 coding

Step 1: Draw up whole data path shown in Fig. 5. There are five OPs and two MUXs in the whole data path.

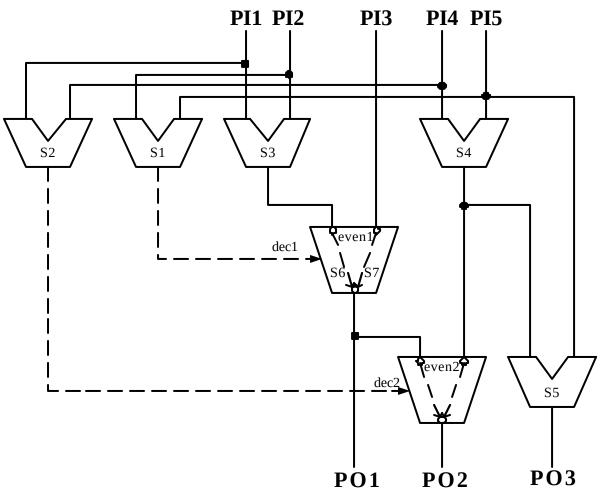
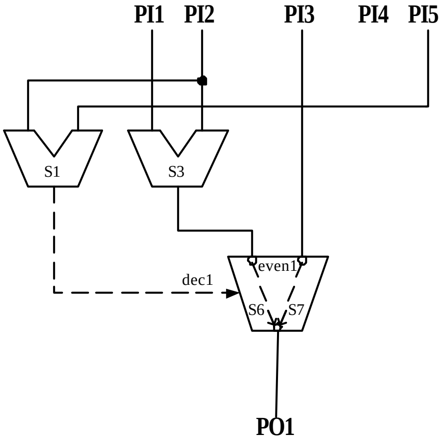


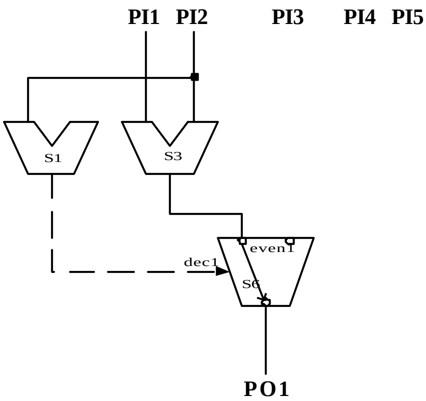
Figure 5. Whole data path



Step 2: Search for mismatches PO, and back-trace form PO to PI. In Fig.4, the PO1 is an error occurs. So we can search out mismatches primary output and error candidates shown in Fig.6.

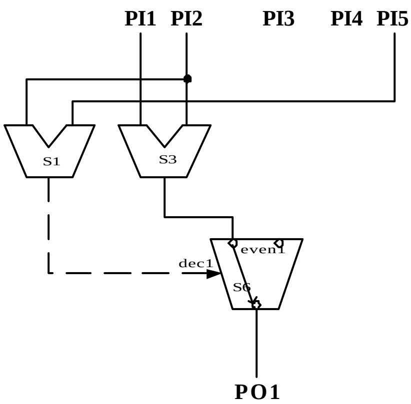
Figure 6. Error occurring of PO1

Step 3: Executed statements of error occurring clock cycle, in which an error appears for the first time to search for variation PI. In Fig. 4, the simulation value and the expected value are not corresponding, so have an error occurs at PO1 at 30ns. At time=30ns, the value change of PI1 and PI2.



Step 4: The corresponding values of PIs will remove in data path show in Fig. 7. The step keeps two OPs and one MUX in the data path.

Figure 7. Remove corresponding values of PIs.



Step 5: Search for incomplete OPs and make up it in data path. In Fig. 7, the S1 is incomplete OP, so make up it. Finally, the final data path is show in Fig.8.

Figure 8. Final data path at PO1.

In this example, the set of statements {even1,dec1,S1, S6,S3} is our error space at PO1 (Fig. 8.).The same as above, the simulation values and the expect values are not corresponding at PO2 at 35ns, so the set of statements {even2,dec2,S2,even1,dec1,S1,S6,S3} is error space at PO2 (Fig. 9).

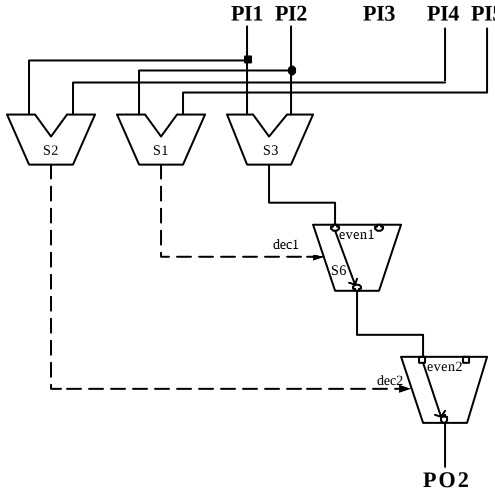


Figure 9. Final data path at PO2.

3.2 Debugging Priority

In this section use a scheme to display the statements in error space with a priority, such that the most suspected statements are reported first. By estimating the probability of correctness for all statements in error space, debugging priority can be calculated for debugging purpose.

This section approach is based on the back-traced technique. The first back-tracing origin is the erroneous primary output. Sometimes there are more than one erroneous primary output at the error-occurring clock cycle. In this situation, every erroneous primary output can be used to construct its own error space. We will discuss some possible manners to construct the error space and propose our strategies to this situation.

We assume that erroneous primary output one (EPO1) and erroneous primary output two (EPO2) are two erroneous primary outputs at the error-occurring clock cycle. The error source of EPO1 is e1 and the error source of EPO2 is e2. Fig.10 shows the possible relationship between e1, e2, EPO1, and EPO2. The triangle region extended by one erroneous primary output denotes its error space and "•"denotes an error source.

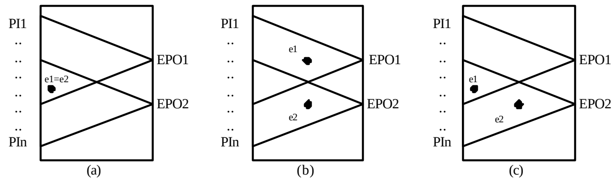


Figure 10. The relationship between errors and EPOs

In Fig.10(a), e1 and e2 are both in the intersection of EPO1 and EPO2. In Fig.10(b), there is no error in the intersection of EPO1 and EPO2. In Fig.10(c), show that both errors are in the error space of one EPO but only one error in the error space of another EPO.

There are three manners to construct the error space with multiple erroneous primary outputs. The first one is to get the union of all individual error space. The error space constructed by this manner may consist of the most errors and the size of it may be larger. The second one is to get the intersection of each individual error space. It is efficient in getting smaller error space, as shown in Fig.10(a), but it is possible to get the wrong error space without any error source in it as shown in Fig.10(b). The third manner, our strategy, is to select one of the erroneous primary outputs to construct the error space. Therefore, the error space size may be much smaller than that of the first manner, and larger than that of the second one.

We continue the example shown in Fig. 3. We find that error-occurring clock cycle is from time=25ns to time=35ns, and have tow error space at PO1 and PO2. First error space of PO1 is {even1,dec1,S1,S6,S3}. Second error space of PO2 is {even1,dec1,S1,S6,S3,even2,dec2, S2}. According to the above three manners, we select one of the EPO2 to construct the final error space.

The final error space is {even1,dec1,S1,S6,S3,even2, dec2,S2}. The approach definition the priority of operational was larger than the priority of multiplexor. In Fig. 9, can obtain priority by conducting a back-trace from PO2 to the PIs according to the relationship in the final data path. According to the above definition, the result calculations show in Fig.11. A statement with less score is displayed first for its high probability to be erroneous.

```
(1)S3      assign   r1 = PI1 & PI2;            High
(1)S6      assign   PO1 = r1;
(2)even1   always @(sel1 or r1 or PI3)
(2)dec1    if (sel1) ... else ...                     Priority
(2)S1      assign   sel1 = PI2 ^ PI5;
(3)even2,
   dec2    assign   PO2=(sel2)?PO1:r2;
(3)S2      assign  sel2=PI1&PI4;                Low
```

Figure 11. The calculation with priority

In the above example, users can see the error statement not only in the error space but also displayed in the priority (1) show in Fig. 11. Therefore, although the number of statements in the error space is eight, users can find their design error at the priority (1) in the display. This paper's method for quick and correct to find design error is practicable.

4. Experimental Results

In this section, we will show the experimental result on four designs written in Verilog coding. Table 1 shows the characteristics of these design circuit. Columns "#Line", "#PI", "#PO", "#MUX" and "#OP" denote the numbers of lines in the HDL, PIs, POs, MUXs and operational modules respectively. The design CM42 is SIS standard benchmark circuit. The design EXM1 is small combinational circuit. The design AVG_4bit is a design for 4bit average value production. The design PCPU is a simple 16-bit pipelined CPU.

Table 2 show is experimental results. The column "#total cases" is error-occurring the number of time. The number of statements in error space is recorded in the column "#Error space Max/Min". In the column "#Case", we report the number of case that the true erroneous appears for each period in the display list of error space.

According to this paper proposed diagnosis method. In Table 2, our method cans estimation of the probability of correctness for each potential error candidate is accurate.

Circuit	#Line	Whole data path			
		#PI	#PO	#MUX	#OP
CM42	52	4	10	0	13
EXM1	65	5	2	2	4
AVG_4bit	91	4	1	0	11
PCPU	159	2	1	6	18

Table 1 Circuit characteristics

Circuit	#total cases	Final data path				#Error space Max/Min	#Case	
		#PI Max/Min	#PO Max/Min	#MUX Max/Min	#OP Max/Min		~30 %	30% ~
CM42	10	4/3	2/1	0/0	3/2	4/2	10	0
EXM1	10	4/4	1/1	2/1	4/3	10/4	8	2
AVG_4bit	10	4/4	1/1	0/0	11/11	11/11	6	4
PCPU	10	2/2	1/1	5/3	6/3	13/8	8	2

Table 2 Results of the design error diagnosis

5. Conclusions

In this paper, an effective algorithm for hierarchically diagnosing RTL circuits is proposed. For the error candidates use simple data path to search for some impossible statements in HDL coding. The estimation of the probability of correctness for each potential error candidates in error space is conducted by data path. Finally, our goal is to minimize the size of error space and the true design errors are always included in. Additionally, we plan to display the statements in error space with a simple priority such that users can quick find their design error in HDL coding.

6. References

[1] V. Boppana, I. Ghosh, R. Mukherjee, J. Jain and M. Fujita, "Hierarchical error diagnosis targeting RTL circuit", In Intl. Conference on VLSI Design, 2000, PP.436-411.

[2] M. Khalil, Y. Traon, and C. Robach, "Towards an Automatic Diagnosis for High-level Validation", In proceeding Intl. Test Conference, 1998, pp. 1010-1018.

[3] T.Y. Jiang, C.N. Jimmy, and J.Y. Jou, "Effective Error Diagnosis for RTL Designs in HDLs", VLSI/CAD Symposium, 2002, pp. 187-190.

[4] H. Wada, T. Masuzawa, K. K. Saluja, and H. Fujiwara, "Design for strong testability of RTL data paths to provide complete fault efficiency", In proceeding Intl. Conference on VLSI Design, 2000, pp 300-305.

[5] T. Masuzawa, M. Izutsu, H. wada, and H. Fujiwara, "Single-control testability of RTL data paths for BIST", In proc. Ninth Asian Test Symposium, 2000, pp.210-215.

[6] K. Yamaguchi, H. Wada, T. Masuzawa, and H. Fujiwara, "A BIST Method Based on Concurrent Single-Control Testability of RTL Data Paths", In proc. Tenth Asian Test Symposium, 2001, pp.313-318.

[7] Petra Michel, Ulrich Lauther and Peter Duzy, "The synthesis approach to digital system design", Kluwer academic publishers-group, Dordrecht, 1992.

[8] D. W. Hoffmann and T. Kropf, "Efficient Design error correction of digital circuits", in Intl. Conference on Computer Design,2000, pp.465-472.

[9] V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and P. Bollineni, "Multiple Error Diagnosis Based on Xlists", in Proc. Design Automation Conf., June 1999, pp.660-665.