Verification on Port Connections

Geeng-Wei Lee*, Chun-Yao Wang✦, Juinn-Dar Huang*, and Jing-Yang Jou*

* Department of Electronics Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.
{gwlee, jyjou}@eda.ee.nctu.edu.tw, jdhuang@mail.nctu.edu.tw

✦ Department of Computer Science
National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
wcyao@cs.nthu.edu.tw

Abstract

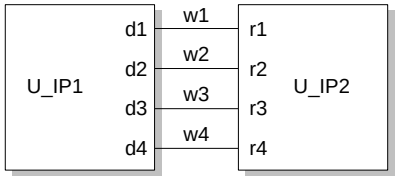
In a system-on-a-chip (SOC) design, several to hundreds of design blocks or intellectual properties (IPs) are integrated to form a complex function. Prior to verify the functionality of the integrated IPs, it is very important to ensure the correctness of the port connections among these IPs. This paper addresses the problem of verification on port connections while IPs are integrated into a larger block or a system, and presents a new connection model and the corresponding error model for port connections. An algorithm providing the minimum pattern set and a general verification flow used to verify port connections are also proposed.

1. Introduction

Traditional interconnect testing for multichip module (MCM) and printed circuit board (PCB) detects and diagnoses the existence of physical defects on the interconnections. The possible faults caused by physical defects are open, short, and stuck faults. Many papers [1-8, 10] tried to find various test sets to detect and diagnose possible faults on an interconnect network. Through boundary scan or physical probes, test patterns can be applied at drivers and response patterns can be observed at receivers on interconnections. By analyzing the observed pattern at receivers, faults can be detected or diagnosed depending on the ability of the applied test patterns.

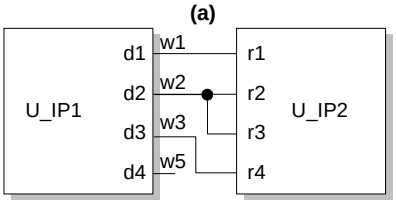
In this paper, we address a slightly different scenario from the traditional interconnect testing: verify the correctness of interconnections of all components in a design. That is, while interconnect testing tries to verify interconnections formed by physical wiring networks, we try to verify interconnections formed by port specifications. And instead of detecting faults that are caused by defects introduced in the process of manufacturing, we try to find out errors in a design that are caused by EDA tools or designers at higher abstract levels. Since the terms: IP, Virtual Component (VC), and design block, are used interchangeably at higher abstract levels in literatures, they are also used interchangeably in this paper.

After the specification and architecture of a design is decided, designers know what IP blocks are going to be integrated in the design. Before starting to verify the whole system by simulation, all necessary IP blocks are connected together by EDA tools automatically or by designers manually. Either way we could have mis-connected components because of carelessness or mis-understanding on port definitions of IP blocks. For convenience, we use Verilog HDL for our examples in this paper. However, our work can not only be applied to Verilog HDL but also to all other hardware description languages (HDLs) which allow bit-precision descriptions of port connections under our assumptions.




```
IP1 U_IP1(.d1(w1),.d2(w2),.d3(w3),.d4(w4));
IP2 U_IP2(.r1(w1),.r2(w2),.r3(w3),.r4(w4));
```

(a)




```
IP1 U_IP1(.d1(w1),.d2(w2),.d3(w4),.d4(w5));
IP2 U_IP2(.r1(w1),.r2(w2),.r3(w2),.r4(w4));
```

(b)

Figure 1. An example of port connection error in Verilog HDL: (a) error-free port connections, and (b) erroneous port connections.

Figure 1 shows a mis-connected example for designs written in Verilog HDL. Figure 1(a) shows correct port connections between two IPs, and Figure 1(b) shows possible mis-connected port connections. We call the errors that lead to mis-connections as "*Port Connection Errors* (PCEs)".

Furthermore, designs may be re-configured during the design exploration process. For example, new IPs may be added for performance enhancement and existing IPs may be removed for cost reduction. And thus port connections in a design are modified, so that PCEs would possibly occur repeatedly in the design process.

While the number of IPs integrated into a system increases, the number of interconnects increases intensively as well and greatly raises the possibility of the occurrence of PCEs.

This paper presents an approach that can verify the correctness of the interconnections among IPs and thus can smooth the verification process of the entire system. A verification flow is also proposed to verify port connection easily and fast, and thus reduces the efforts required to find PCEs from hours to minutes or even seconds.

Following sections are organized as follows. In Section 2, we first review previous work on interconnect testing, and some necessary definitions, notations, and models are also described. Verification patterns and their diagnosablilty are discussed in Section 3. The proposed verification flow is described in Section 4. Section 5 concludes this paper.

2. Preliminary

2.1 Previous Work

Lien and Breuer [5], Shi and Fuchs [8] defined different levels of *diagnostic resolution* (DR) to assess the diagnosability of a test pattern set. The diagnostic resolutions range from the lowest resolution of determining if an interconnect is fault-free to the highest resolution of identifying all faults in an interconnect.

Fault models used in the previous work on interconnect testing are primarily based on one of the following three fault models: (1) *short fault only*, (2) *short and stuck fault*, and (3) *short, stuck, and open faults*. Since it is more realistic to take all short, stuck, and open faults into consideration, the third fault model is commonly used and discussed in the latest work on interconnect testing.

Also, interconnects can be *behavioral* or *structural*. No adjacency information of nets is available in a behavioral interconnect while some or detailed adjacency information is available in a structural interconnect. For structural interconnects, test patterns can be further reduced by exploiting available structural information [1, 5, 6, 10].

To diagnose an interconnect, there are *non-adaptive* or *adaptive* methods. In a non-adaptive diagnosis method, diagnosis starts only after all patterns are applied. In an adaptive diagnosis method, diagnosis starts after a group of leading patterns is applied and the corresponding response may affect the succeeding patterns applied. Non-adaptive diagnosis is also known as *one-step* diagnosis, and adaptive diagnosis is also known as *two-step* diagnosis in some papers. Adaptive diagnosis methods require more computations, but usually can reduce test patterns for diagnosis[5, 8]. Previous work on interconnect testing primarily focuses on finding minimum test patterns to diagnose an interconnect.

Since our work in this paper targets on verifying designs at abstract levels higher than gate-level, we assume that structural information is not available (It is true for most of designs.) and ports of IPs are *behaviorally* connected. To prevent from getting confused with the definition of “fault” used in testing, we use “error” that is commonly used in the verification field to present incorrect connections. And the term “port connections” we use in this paper is analogous to the “interconnect” in interconnect testing. Port connections among IPs at higher abstract levels form physical interconnects in a physical design.

Compare our work with the similar work in [9] that verifies port connections only based on the *port-order fault model*, we verify port connections based on a more general error model that can model all possible connection errors including the errors identical to *port-order faults*.

For simplicity, we only define two levels of diagnostic resolution that are further simplified from those defined in the previous work [5, 8]:

- DR1: Determine if port connections are error-free.
- DR2: Identify all errors in port connections.

2.2 Port Connection Model (PCM) and Definitions

All connection ports can be divided into two groups, *driver group* and *receiver group*. Input ports of all design blocks are regarded as drivers, and output ports are regarded as receivers. For inout ports that can be either input or output ports, they can be regarded as either drivers or receivers. To generalize all port connections, we use a *port connection model* (PCM) shown in Figure 2 that models port connections between drivers and receivers.

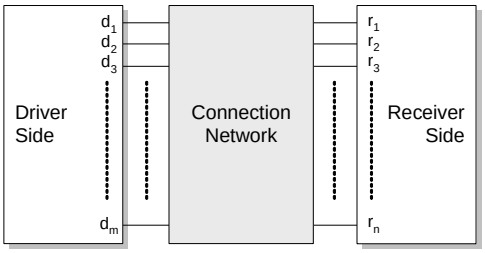


Figure 2. Port connection model.

Given an $I_{mn}(D,R,W)$ that presents port connections, we have:

- Driver set D:
  $D=\{d_1,d_2,...,d_m\}$, $|D| = m$: *number of drivers in driver set.*
- Receiver set R:
  $R=\{r_1,r_2,...,r_n\}$, $|R| = n$: *number of receivers in receiver set.*
- Net (Wire) set W:
  $W=\{w_1,w_2,...,w_l\}$, $|W| = l$: *number of nets,*
  $l \le \min(m,n)$.

We further define following relationships:

- $R(d_i)$: Receivers of a driver d_i,
  $R(d_i)=\{r_{i1},r_{i2},...,r_{ij} \mid r_{i1},r_{i2},...,r_{ij} \in R,\ j \ge 1\}$
- $D(r_i)$: Drivers of a receiver r_i,
  $D(r_i)=\{d_{i1},d_{i2},...,d_{ik} \mid d_{i1},d_{i2},...,d_{ik} \in D, k \ge 1\}$
- $D(w_i)$: Drivers on a net w_i,
  $D(w_i)=\{d_{i1},d_{i2},...,d_{ix} \mid d_{i1},d_{i2},...,d_{ix} \in D, x \ge 1\}$,
  $|D(w_i)| = x$: *number of drivers on* w_i.

  $R(w_i)$: Receivers on a net w_i,
  $R(w_i)=\{r_{i1},r_{i2},...,r_{iy} \mid r_{i1},r_{i2},...,r_{iy} \in R, y \ge 1\}$,
  $|R(w_i)| = y$: *number of receivers on* w_i.
- $C(w_i)$: Drivers and receivers on a net w_i,
  $C(w_i) = D(w_i) \cup R(w_i)$
  And for each net on $I_{mn}(D,R,W)$, we also have:
  $C(w_1) \cup C(w_2) \cup ... \cup C(w_l) = D \cup R$
  $C(w_i) \cap C(w_j) = \phi;\ w_i, w_j \in W, i \ne j.$
  Also, w_i is called a *simple net* if $|D(w_i)|=1$ and $|R(w_i)|=1$, i.e., exact one driver and one receiver are connected to it. And w_i is called a *complex net* if $|D(w_i)|>1$ or $|R(w_i)|>1$, i.e., more than two drivers or receivers are connected to it. A complex net w_i is said a *multiple-drive* net if $|D(w_i)| > 1$, and said a *multiple-fanout* net if $|R(w_i)| > 1$. A complex net could be multiple-drive, multiple-fanout, or both.

  Let N_c = *number of complex nets* in an interconnect and N_s = *number of simple nets* in an interconnect, we have $|W| = l = N_c+N_s$.

If a port is connected to more than one other ports in a design, there must exist at least one complex net. It is usually the case that if a design contains multiple-fanout ports or tri-state buses.

For convenience, we repeat some notations and definitions established in [2] as follows:

- **Parallel Test Vector** (PTV): the vector applied to all drivers in parallel.
- **Sequential Test Vector** (STV): the vector applied to a driver in serial.
- **Verification Pattern Set** (S): the collection of all STVs, S = {STV_1, STV_2, …, STV_m}. Each STV may have different bit length, and the bit length of the longest STV is the number of (PTV) patterns required.
- **Sequential Response Vector** (SRV): the response vector observed at receivers. An SRV can be a vector contributed by one or more STVs. For any SRV that is contributed by multiple STVs, the value in its vector is a result of certain logic operations of all contributing STVs.
- **Response Pattern Set** (S'): the collection of all SRVs, S' = {SRV_1, SRV_2, …, SRV_n}
- **Syndrome**: the SRV of a connection error.
- **Aliasing syndrome**: the resulting syndrome of a set of erroneous nets is the same as a correct SRV of a net not in the set.
- **Confounding Syndrome**: identical syndromes that result from different sets of multiple independent errors.

Figure 3 shows an example of a verification pattern set. The pattern set consists of six patterns that applied to four drivers. Note the shaded patterns are shown for clarity to denote PTV and STV respectively.

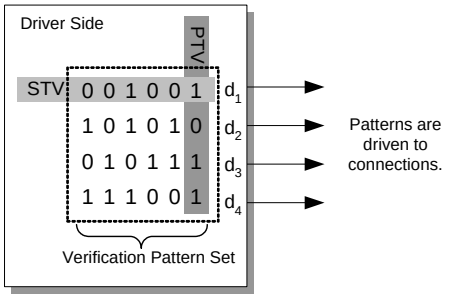


Figure 3. An example of a verification pattern set.

2.3 Assumptions

Unlike the environment in the previous work on interconnect testing, we verify the correctness of port connections of a design at a higher abstract level. Since there are various types of EDA (Electronic Design Automation) tools and environments for designers, we make some assumptions described in the following to make our work viable:

(1) *The simulation environment allows 4-value (0-1-X-Z) logic simulation:* Most simulators nowadays support 4-value logic simulation.

(2) *Output ports that are supposed not to be connected to anything are discarded*: In some cases, some output ports of IPs are not in use in certain configuration and not connected to anything in the design. These ports should be removed first before the verification to avoid false negative alarms.

(3) *A net that is driven by different logic value 0 and 1 is supposed to have logic value X (unknown):* It is true for most simulators that allow 4-value simulation.

(4) *For inout ports, either input or output direction must be specified before the verification and not changeable during the verification process:* Note that we are verifying the correctness of the port connections, the direction of ports does not matter.

(5) *For multiple-drive and multiple-fanout nets, assumptions are made as follows. (i) Repeaters or bus keepers used to hold logic values on nets are removed. (ii) No wired-logic behavior is assumed. (iii) No driving strength is assumed:* Information about driving strengths is usually not available for designs at abstract levels higher than gate-level. Even for a design that having driving strength information, it is not hard to temporally remove it.

(6) *For a net that is not connecting to any driver in the driver set, it would have a fixed logic value (0, 1, or X) during the verification process*: A net that is not connecting to any driver in the driver set is either connected to no drivers or connected to drivers providing fixed logic values during the verification process.

The assumptions made above are mostly self-satisfied in most EDA environments for designs at a abstract level higher than gate-level.

2.4 PCE Model

We categorize PCEs into two categories, *floating errors* and *connection errors*. Based on the PCM in Section 2.2, these two categories of PCEs are described as follows:

- **Floating errors**:
  A driver $d_i \in D$ is floating if *no* receiver receives its value, and a receiver is floating if *no* driver drives it.
- **Connection errors**:
  A driver $d_i \in D$ is said to have erroneous connections if it is connected to any receiver r_j that $r_j \notin R(d_i)$. A receiver $r_i \in R$ is said to have erroneous connections if it is connected to any driver d_j that $d_j \notin D(r_i)$.

To distinguish our work from traditional interconnect testing, we use some different terminology but basically presenting the same ideas as in testing. A *floating error* is like an open fault in testing, and a *connection error* is like a short fault or a combination of several short and open faults.

3. Verification Patterns

In this section, we first discuss the algorithm deriving verification patterns with the simplified PCM that is analogous to the interconnect model commonly used in interconnect testing. Then the algorithm is extended to handle the PCM.

Since we verify the port connections by simulation, the number of verification patterns is not as critical as that in traditional interconnect testing. In interconnect testing, number of test patterns presented in literatures so far requires at least $O(\log n)$[1] and at most $O(n)$ patterns [8], where n is the number of nets on an interconnect. The difference for the time to simulate the least and the most patterns is possibly less than seconds and at virtually no cost. But we still present a method to generate the minimum number of patterns to verify port connections.

3.1 n-to-n Port Connections with All Simple Nets

Given $I_{nn}(D,R,W)$ of error-free port connections simplified from the PCM with 2 extra constraints: *(1) m=n, (2) d(wi)=r(wi)=1, for all* $w_i \in W$*, i.e., all nets are simple nets*. This is also the model that is commonly used in interconnect testing. Neither multiple-drive nor multiple-fanout nets are allowed.

In [8], $\lceil \log(n+2) \rceil$ test patterns had been proved to be necessary and sufficient for reaching the lowest diagnosis resolution, i.e., DR1. And it is also the lower bound of the number of required patterns in interconnect testing that takes all fault models into consideration.

Using the PCM and assumptions in Section 2, we can use almost as less as the lower bound of patterns in interconnect testing to diagnose port connections in $I_{nn}(D,R,W)$ but reach DR2 instead of DR1.

[1] For convenience, $\log_2$ is denoted as $\log$ in this paper.

Theorem 1: To diagnose all errors in $I_{nn}(D,R,W)$, i.e., to reach DR2, $\lceil \log(n+1) \rceil + 1$ verification patterns are necessary and sufficient.

Proof:
Necessity:
(1) To identify each and every driver, a unique bit string (STV) is required for each and every driver.
⇨ $\lceil \log(n) \rceil$ patterns are required.
(2) To detect any receiver that receives a fixed logic value 0 or 1, neither all-0 nor all-1 STV is allowed. A receiver that receives a fixed logic value is identified as a floating error.
⇨ $\lceil \log(n+2) \rceil$ patterns are required.
(3) To avoid confounding syndromes that all receivers are floating and all ports are connected (both cases lead to an all-X value in SRVs), an extra all-0 or all-1 PTV is required. Note that this pattern can further reduce $\lceil \log(n+2) \rceil$ to $\lceil \log(n+1) \rceil$ in (2), since it can avoid either all-1 or all-0 STVs.

In summary, $\lceil \log(n+1) \rceil + 1$ patterns are necessary.

Sufficiency:
After simulation, if a receiver r_i receives $SRV_{if} \neq SRV_i$, an error is detected and we can further identify errors by analyzing SRV_{if} :
(1) SRV_{if} is all-X: Receiver r_i is floating. If not, the all-0 or all-1 PTV guarantees at least one bit of SRV_{if} is not *X*.
(2) *One or more bits in* SRV_{if} *are X*: Since each STV is unique, only cases that two or more drivers drive the receiver r_i would generate *X* in SRV_{if}. Furthermore, we can identify which drivers are involved in the errors by analyzing bit positions of value *X* in SRV_{if}.
(3) *No bits in* SRV_{if} *is X*: The receiver r_i is mis-connected to a wrong driver.

Also, a floating driver d_j can be diagnosed by analyzing the verification pattern set since no contribution can be found by the corresponding STV_j.

Neither aliasing syndromes nor confounding syndromes can occur since a receiver would receive $SRV_{if} \neq SRV_i$ if there exists any PCEs in $I_{nn}(D,R,W)$ by analyzing the response set at the receivers. ■

Table 1 shows an example of the verification patterns for a 4-to-4 port connections where all nets are simple nets. The pattern in the shaded column is the extra all-0 PTV, and the others are generated by the *counting algorithm* without all-0 and all-1 STVs. Next, we extend port connections to the general PCM.

Table 1. An example of verification patterns.

Driver	Verification Pattern (STV)	
d_1	0	001
d_2	0	010
d_3	0	011
d_4	0	100

3.2 m-to-n Generalized Port Connections

Given $I_{mn}(D,R,W)$ of error-free port connections, where *no constraints are set*, it presents a more realistic and general case of port connections. That is, the existence of complex nets is allowed. To verify such port connections, two phases of verification are required. The first phase verifies the connections as n-to-n port connections without complex nets, and the second phase ensures that all complex nets are properly connected as expected. These two verification phases are described as follows:

- *Phase 1*:
  Regard all complex nets in $I_{mn}(D,R,W)$ as simple nets, and $I_{mn}(D,R,W)$ effectively becomes *x-port to x-port* connections, where $x=|W|$. Use the verification patterns discussed in Section 3.1 to verify such connections, we need ($\lceil \log(|W|+1) \rceil + 1$) = ($\lceil \log(N_s + N_c + 1) \rceil + 1$) patterns. For a complex net w_i, all drivers in $D(w_i)$ are regarded as a single driver and drive the same STV, meanwhile, all receivers in $R(w_i)$ are regarded as a single receiver and should receive the same SRV. *Phase 1* can diagnose all port connection errors except ones that not all drivers on a complex net are floating, since all drivers on a complex net are driving same STV and thus the responses are not distinguishable. For example, for a complex net w_1 that $C(w_1) = \{d_1, d_2, d_3, r_1\}$, if only d_1 is floating, it can not be detected since r_1 still receives a correct SRV.
  Phase 1 detects and diagnoses all floating and connection errors on all receivers, and all connection errors on all drivers.

- *Phase 2*:
  Verify the connectivity of drivers on all complex nets. Note that for a complex net w_i, the connectivity of all receivers in $R(w_i)$ is ensured in *Phase 1* if they all receive the same SRV. To ensure the connectivity of all drivers in $D(w_i)$, $|D(w_i)|$ patterns that generated by *walking-one* or *walking-zero* method are required (proved later). Therefore, to verify $I_{mn}(D,R,W)$ in *Phase2*, number of verification patterns required is $max(|D(w_i)|)$, where $\{w_i \in W, i = 1,2,...,|W|\}$. And all receivers on complex nets are expected to receive the SRV of *all-X* value.
  Phase 2 detects and diagnoses all floating errors on all drivers.

Theorem 2: To verify the connectivity of *n* drivers on a complex net, a pattern set must have at least *n* different patterns of *one-hot* or *one-cold* vectors.

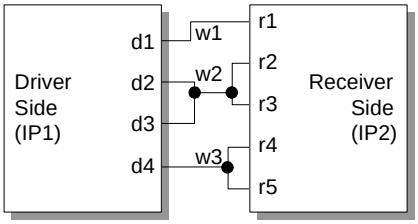
Proof:
A *one-hot vector* is a vector with only one bit of value 1 and all other bits of value 0, and a *one-cold vector* is a vector with only one bit of value 0 and all other bits of value 1. For a pattern set $S = \{PTV_1, PTV_2, \ldots, PTV_x\}$ that is used to verify the connectivity of drivers $\{d_1, d_2, \ldots, d_n\}$ on a complex net, each PTV of value $\{v_1, v_2, \ldots, v_n\}$ is applied to all drivers in the manner of $\{v_1 \rightarrow d_1, v_2 \rightarrow d_2, \ldots, v_n \rightarrow d_n\}$. To verify if a driver d_i is connected on the complex net, a PTV must be applied to all drivers that the bit value of v_i applied at d_i is different from all other bits in the PTV, i.e., a vector with $v_i=0$ and all others are ones (*one-cold vector*), or $v_i=1$ and all others are zeros (*one-hot vector*), so that only d_i can possibly contribute the *X* value to receivers. Therefore, *n* drivers on a complex net require at least *n* different patterns of *one-hot* or *one-cold* vectors to verify the connectivity, i.e., to ensure that all drivers are indeed connected to the complex net. ■

By Theorem 2, it is obvious that the minimum number of patterns required to verify the connectivity of drivers on a complex net is the number of drivers on it. Therefore, number of patterns required to verify all complex nets in *Phase 2* is determined by the complex net that has the most drivers. The simplest way to generate such patterns is applying *walking-one* or *walking-zero* sequences.

In summary, to verify $I_{mn}(D,R,W)$, we need a total number of $(\lceil \log(|W|+1) \rceil + 1 + max|D(w_i)|)$ to reach DR2.

Since the term $(max|D(w_i)|)$ is the number of additional patterns required in m-to-n port connections compared to n-to-n port connections, verification patterns can be reduced if $(max|D(w_i)|)$ can be reduced. Therefore, for a complex net having ports that can be either drivers or receivers, i.e., *inout ports*, patterns can be minimized if inout ports are configured as *output ports* as many as possible. But note that at least one driver must be left on a complex net for the use of pattern application.

3.3 An Example




```
IP1 U_IP1(.d1(w1),.d2(w2),.d3(w2),.d4(w3));
IP2 U_IP2(.r1(w1),.r2(w2),.r3(w2),.r4(w3),.r5(w5));
```

Figure 4. An example of possible port connections.

Figure 4 shows an example of 4-to-5 port connections.

To derive verification patterns discussed in Section 3.2, total number of verification patterns required is calculated as: $\lceil \log(3+1) \rceil + 1 + 2 = 5$. And these five verification patterns are shown in Table 2. Note that the verification patterns required in *Phase 1* and *Phase 2* are listed in the separate columns in Table 2. In *Phase 1*, d_2 and d_3 apply the same STV (shown in the shaded area), because they are both on the same complex net (w_2).

Table 2. Verification patterns for the example in Figure 4.

Driver	Verification Pattern (STV)	
	Phase 1	Phase 2
d_1	001	
d_2	010	01
d_3	010	10
d_4	011	

After the verification patterns are applied at the driver side, expected response patterns at the receiver side are listed in Table 3. Note that the SRVs containing *all-X* values are expected at r_2 and r_3 (shown in the shaded area), because they are both on the same complex net (w_2).

Table 3. Response patterns for the example in Figure 4.

Receiver	Response Pattern (SRV)	
	Phase 1	Phase 2
r_1	001	
r_2	010	XX
r_3	010	XX
r_4	011	
r_5	011	

4. Verification Flow

Like most verification work, design intent is virtually impossible to be thoroughly verified and so is the intent of *correct* port connections. We exploit redundancy like many of other verification methods to reduce the possibility of errors. That is, a design can be described in two different formats, and the correctness of the design can be ensured by checking whether they are consistent.

The verification flow we propose here also requires two formats of port connections. One is the design written in any HDL like Verilog HDL or VHDL, etc., and the other is the format that purely describes the connections of all ports and it can be as simple as a two-column table. Drivers in port connections are described in one column, and corresponding receivers are described in the other.

Figure 5 shows our verification flow. Once the design specification is decided, designers start HDL coding for the design; meanwhile, a file is maintained for port connections descriptions. Processes in the shaded area shown in Figure 5 can be done automatically. Verification

and Response patterns are generated from the file with the information of port connections. A testbench can be automatically generated from the whole design written in HDL that will be used for simulation later. In the testbench that is automatically generated, all modules are the *stub models* that only have the information of port interfaces extracted from the HDL design, and an additional process that is used to apply verification patterns and analyze response patterns is also added.

After simulating the testbench, a *pass* signal is asserted if port connections are correct. Otherwise a *fail* signal is asserted and all PCEs in port connections would be reported. Designers can verify the port connections based on the information reported and go through the verification flow again whenever the port connections are modified.

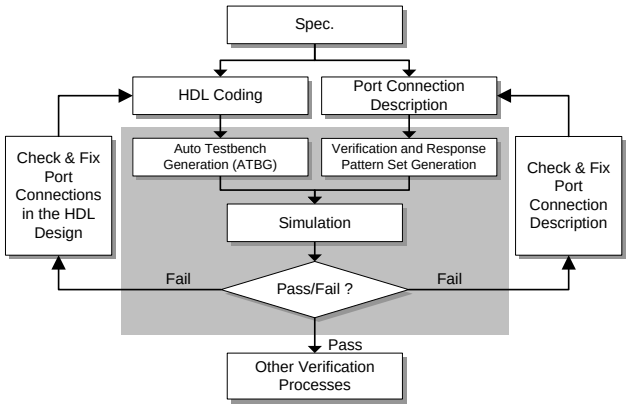


Figure 5. Flow Chart of the Verification Flow

Note that both descriptions for port connections could be erroneous, but it is indeed a fundamental problem in verification: for a design written in two different formats or even by two different teams, neither one is guaranteed a golden description of the design. But the possibility of errors can be greatly reduced by exploiting such redundancy.

5. Conclusions

Before verifying the function of a system, designers have to connect all modules together for simulation. Though it is not a hard work to verify the correctness of port connections manually, it is extremely time-consuming. It is especially true when SOC designs are getting more complex and designers have to put more IPs into one system that have thousands of ports to be connected. Also, configurations of a design may be changed from time to time to accommodate performance requirement during the design exploration process. This process leads to the iterative modification of port connections. It also means that errors in port connections could occur repeatedly and must be verified whenever port connections are modified.

In this paper, we present a port connection model (PCM) to model port connections at higher abstract levels, and a port connection error (PCE) model to describe all possible errors in port connections. By taking advantage of most simulators that support 4-value logic simulation, we propose an algorithm generating a minimum verification pattern set and a verification flow that can diagnose all PCEs in port connections efficiently. We believe that this work is very practical and can save lots of time compared with the time required to verify port connections manually, say, from hours to fewer than minutes or even seconds.

6. References

[1] W.-T. Cheng, J. L. Lewandowski, and E. Wu, "Optimal diagnostic methods for wiring interconnects," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, Sep. 1992, pp. 1161-1166.

[2] A. Hassan, J. Rajski, and V. K. Agarwal, "Testing and diagnosis of interconnects using boundary scan architecture," in *Proc. IEEE International Test Conference*, 1988, pp. 126-137.

[3] Najmi Jarwala and C. W. Yau, "A new framework for analyzing test generation and diagnosis algorithms for wiring interconnect," in *Proc. IEEE International Test Conference*,1989, pp.63-70.

[4] W. H. Kautz, "Testing for faults in wiring networks," *IEEE Transaction on Computer*, vol. C-23, Apr.1973, pp. 358-363.

[5] J.-C. Lien and M. A. Breuer, "Maximal diagnosis for wiring networks," in *Proc. IEEE International Test Conference*, 1991, pp.96-105.

[6] D. McBean and W. R. Moore, "Testing interconnects: a pin adjacency approach," in *Proc. IEEE European Test Conference*, 1993, pp.484-490.

[7] G. D. Robinson and J. G. Deshayes, "Interconnect testing of boards with partial boundary scan," in *Proc. IEEE International Test Conference*, 1990, pp.572-581.

[8] W. Shi, and W. K. Fuchs, "Optimal interconnect diagnosis of wiring networks," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 3, Sept. 1995, pp. 430-436.

[9] C. Y. Wang, S. W. Tung, and J. Y. Jou, "On automatic verification pattern generation for SOC with port order fault model," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 4, Apr. 2002, pp.466 – 479.

[10] C. W. Yau and N. Jarwala, "A unified theory for designing optimal test generation and diagnosis algorithms for board interconnects," in *Proc. IEEE International Test Conference*, 1989, pp. 71-77.