Verification of Pin-Accurate Port Connections

Geeng-Wei Lee and Juinn-Dar Huang
National Chiao Tung University

Jing-Yang Jou
National Chiao Tung University

Chun-Yao Wang
National Tsing Hua University

Before verifying the functionality of SoCs, designers must ensure the correctness of the pin-accurate interfaces of up to hundreds of integrated IP blocks. This article presents a new connection model and a corresponding error model for pin-accurate port connections, along with an algorithm for generating the minimum pattern set, a methodology for diagnosing errors, and a port connection verification flow.

TRADITIONAL INTERCONNECT TESTING of multichip modules and PCBs detects and diagnoses the existence of physical defects on an interconnect network. Physical defects can cause open, short, and stuck-at faults. Many researchers have developed test sets for detecting and diagnosing these faults in interconnect networks. Using boundary scan or physical probes, designers apply test patterns at drivers and observe response patterns at receivers on the interconnects.[1] By analyzing the observed patterns, they can detect or diagnose faults, depending on the quality of the applied test patterns.

In this article, we address a slightly different scenario from traditional interconnect testing: verifying the correctness of interconnections for all components in a design coded with a hardware description language (HDL). Whereas traditional interconnect testing verifies interconnections formed by physical wiring networks, we verify interconnections formed by port specifications. Instead of detecting faults caused by defects introduced in manufacturing, we try to find errors caused by EDA tools or designers at higher design abstraction levels.

After deciding on the design's specification and architecture, designers determine which IP blocks will be integrated in the design. Designers must refine these IP blocks with pin-accurate interfaces. Before verifying the system through simulation, designers connect all necessary IP blocks, either manually or automatically. Either way, incautious EDA tool use, careless manual HDL coding, or misunderstanding of IP port definitions can result in misconnected components. Connecting ports among IP blocks is error-prone, especially for IP blocks integrated manually.

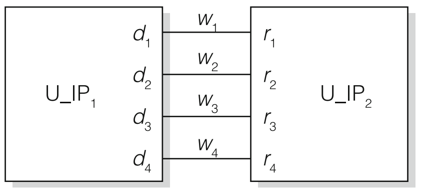
We present a method that verifies the correctness of port connections between IP blocks and thus reduces the verification effort for the entire system. Our main contributions are

- a port connection model for designs with pin-accurate interfaces at higher abstraction levels,
- an algorithm for generating the minimum number of verification patterns,
- an algorithm for resolving the response patterns to achieve high diagnostic resolution, and
- a verification flow for port connections.

Preliminaries

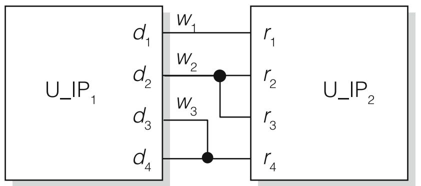
For convenience, we use the Verilog HDL for our examples. However, our work also applies to all other HDLs that allow bit-precision descriptions of port connections. Figure 1 shows a simple example of such a description. In this example, all ports in U_IP1 are output ports (drivers d_1 through d_4), and all ports in U_IP2 are input ports (receivers r_1 through r_4). Figure 1a shows the correct port connections, and Figure 1b is a case of possible misconnected port connections. *Port connection errors* (PCEs) are errors that lead to misconnections.

Assume we apply the patterns shown in Figure 1c to the ports in U_IP1. Figure 1d shows the patterns



```
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(w3),.d4(w3));
IP2 U_IP2(.r1(w1),.r2(w2),.r3(w2),.r4(w3));
```

(b)

d_1	00	r_1	00	r_1	00
d_2	01	r_2	01	r_2	01
d_3	10	r_3	10	r_3	**01**
d_4	11	r_4	11	r_4	**1X**

(c) (d) (e)

Figure 1. Example port connection error in Verilog: error-free port connections and the corresponding HDL code (a); erroneous port connections, and the HDL code shown in bold indicating errors (b); patterns applied at U_IP1 (c); observed patterns at U_IP2 in error-free port connections (d); observed patterns at U_IP2 in erroneous port connections, with bold items indicating the differences (e). (d_1 … d_4: drivers; r_1 … r_4: receivers; U_IP1, U_IP2: modules; w_1 … w_4: wires, X: unknown logic value.)

observed at the ports in U_IP2 after simulation for Figure 1a, and Figure 1e shows them for Figure 1b. Because the port connections in Figure 1a are error free, the observed patterns in Figure 1d are the golden responses. By comparing the observed patterns in Figure 1e with those in Figure 1d, we easily determine that r_3 and r_4 must be involved with PCEs because they have different responses from the golden ones. Also, the response patterns in Figure 1e show that we can observe an unknown logic value (X) at the receivers, assuming we are using a four-valued HDL simulator. This four-valued simulation approach is considerably different from traditional interconnect testing, and we take advantage of it in our work.

Comparison with previous work

Because the problem to be solved in this article is similar to that of interconnect testing, we first discuss previous work on that problem. Many researchers have tried to find the optimal test set for diagnosing an interconnect network.[2–9] For example, Lien and Breuer and Shi and Fuchs define different diagnostic resolution levels to assess a test pattern set's diagnosability.[2,3] These levels range from the lowest resolution (determining whether an interconnect network is fault free) to the highest resolution (identifying all faults). Previous work on interconnect testing primarily uses one of three fault models: short fault only; short and stuck-at faults; or short, stuck-at, and open faults.

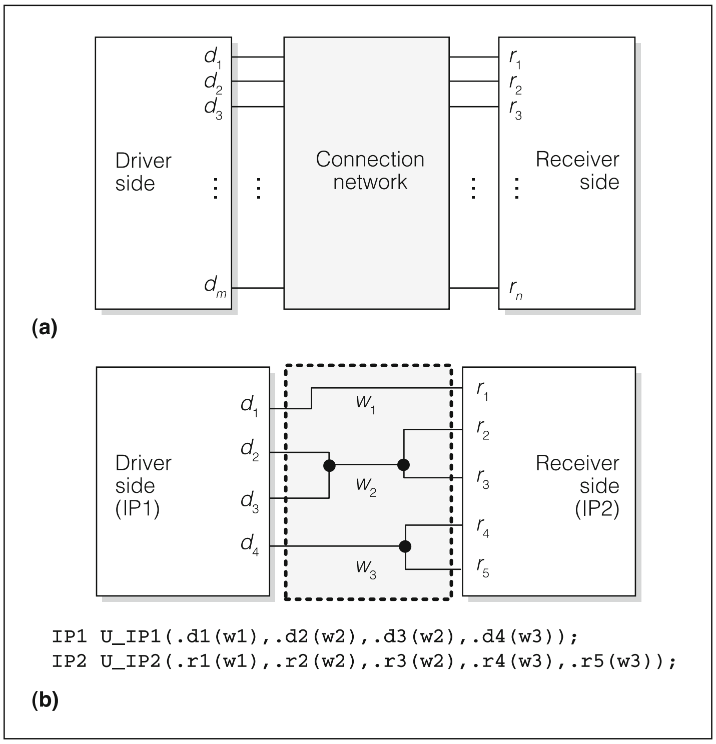
The methods of diagnosing an interconnect network are nonadaptive or adaptive. *Nonadaptive* (or one-step) diagnosis starts only after all patterns are applied.[7] *Adaptive* (or two-step) diagnosis starts after a group of leading patterns is applied, and the corresponding responses can affect the succeeding patterns applied.[2,3] Adaptive diagnosis methods require more computations but can usually reduce the number of test patterns. The most popular and well-defined structure for applying patterns and retrieving responses in interconnect testing is boundary scan.[10,11]

Most previous research on interconnect testing focused on finding the minimum set of test patterns to diagnose an interconnect network. In contrast, Wang, Tung, and Jou propose verifying port connections only, using a port-order fault model.[12] We also verify port connections, but we use a more general error model that represents all possible connection errors, including those identical to port-order faults.

Interconnect testing versus port connection verification

Rather than trying to detect and diagnose faults on physical interconnects, port connection verification tries to verify interconnects formed by port connections in designs coded with HDL that have pin-accurate interfaces. The following are the main differences between interconnect testing and port connection verification:

- *Pattern application and response observation*. In interconnect testing, engineers apply patterns and observe responses either through boundary scan or physical signal probes. Port connection verification requires an HDL simulator. We apply patterns through an HDL-coded testbench pro-



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

Figure 2. Port connection model (a), and an example design and its HDL code (b). (d_1 … d_4: drivers; r_1 … r_5: receivers; U_IP1, U_IP2: modules; w_1, w_3: simple nets; w_2: complex net.)

cess, and we observe responses through another such process or dump them into a file.

- *Logic values*. In interconnect testing, all signal values are either logic 1 or logic 0. However, because we use a four-valued HDL simulator in port connection verification, we can have two additional signal values, X (unknown) and Z (high impedance). We take advantage of this feature to further reduce verification patterns and enhance their diagnosability.
- *Faults and errors*. In interconnect testing, possible faults include a broken wire, an extra wire bridging two wires, or a combination of these physical defects. In port connection verification, errors are induced by designers or tools at higher abstraction levels. Because both types of problems occur on interconnections, the error model in port connection verification shares some properties with the interconnect-testing fault model. (To prevent confusion with the word *fault* used in testing, we use *error*, commonly used in the verification field, to denote incorrect connections.)

Further simplifying previous work, we define only two levels of diagnostic resolution: DR1 (determining if port connections are error free) and DR2 (identifying all errors in port connections).

Assumptions

Because designers use various kinds of EDA tools and environments, we make several assumptions to make our methodology viable:

- The simulation environment allows four-valued (0, 1, X, Z) logic simulation.
- Output ports that should not be connected to anything are discarded.
- A net simultaneously driven by logic values 0 and 1 should have logic value X.
- *Inout* ports must be specified before verification as either input or output, but not both, and are not changeable during the verification process.
- For multiple-drive and multiple-fan-out nets, we make three assumptions: repeaters or bus keepers used to hold logic values on nets are removed; there is no wired-logic behavior; and there is no driving strength.
- A net that is not connected to any driver in the driver set has a fixed logic value (0, 1, X, or Z) during the verification process.

Designs at abstraction levels higher than gate level usually satisfy these assumptions in most EDA environments.

Port connection model and definitions

We categorize all connection ports into two disjoint sets: drivers and receivers. The output ports of all design blocks are drivers, and the input ports are receivers. Ports that can be either input or output ports can be either drivers or receivers, but not both. To generalize all port connections, we use a port connection model (PCM) that models port connections between drivers and receivers. Figure 2 shows the general model and an example design that demonstrates our definitions.

Let $I_{mn}(D, R, W)$ represent m-port to n-port connections for driver set D, receiver set R, and net (wire) set W, which we define as follows:

- Driver set D: $D = \{d_1, d_2, \ldots, d_m\}$, $|D| = m$, the number of drivers in the driver set.
- Receiver set R: $R = \{r_1, r_2 \ldots r_n\}$, $|R| = n$, the number of receivers in the receiver set.

- Net (wire) set W: $W = \{w_1, w_2, \ldots, w_t\}$, $|W| = t$, the number of nets in the connection network, $t \leq \min(m, n)$.

We also define the following terms:

- $R(d_i)$: receivers of driver d_i, $R(d_i) = \{r_{i1}, r_{i2}, \ldots, r_{ij} \mid r_{i1}, r_{i2}, \ldots, r_{ij} \in R, j \geq 1\}$, $|R(d_i)| = j$, the number of receivers driven by d_i.
- $D(r_i)$: drivers of receiver r_i, $D(r_i) = \{d_{i1}, d_{i2}, \ldots, d_{ik} \mid d_{i1}, d_{i2}, \ldots, d_{ik} \in D, k \geq 1\}$, $|D(r_i)| = k$, the number of drivers driving r_i.
- $R(w_i)$: receivers on net w_i, $R(w_i) = \{r_{i1}, r_{i2}, \ldots, r_{iy} \mid r_{i1}, r_{i2}, \ldots, r_{iy} \in R, y \geq 1\}$, $|R(w_i)| = y$, the number of receivers on w_i.
- $D(w_i)$: drivers on net w_i, $D(w_i) = \{d_{i1}, d_{i2}, \ldots, d_{ix} \mid d_{i1}, d_{i2}, \ldots, d_{ix} \in D, x \geq 1\}$, $|D(w_i)| = x$, the number of drivers on w_i.
- $C(w_i)$: drivers and receivers on net w_i, $C(w_i) = D(w_i) \cup R(w_i)$.

For each net on $I_{mn}(D, R, W)$, we also have $C(w_1) \cup C(w_2) \cup \ldots \cup C(w_1) = D \cup R$, and $C(w_i) \cap C(w_j) = \phi$; $w_i, w_j \in W$, $i \neq j$.

We call w_i a simple net if $|D(w_i)| = 1$ and $|R(w_i)| = 1$; that is, exactly one driver and one receiver are connected to it. We call w_i a complex net if $|D(w_i)| > 1$ or $|R(w_i)| > 1$; that is, more than one driver and receiver are connected to it. A complex net w_i is a multiple-drive net if $|D(w_i)| > 1$, and it is a multiple-fan-out net if $|R(w_i)| > 1$. Thus, a complex net can be multiple drive, multiple fan-out, or both. N_c is the number of complex nets, N_s is the number of simple nets in a connection network, and $|W| = t = N_c + N_s$.

If a port is connected to more than one other port, at least one complex net must exist. This is usually the case if a design contains multiple-fan-out ports or tristate buses.

For convenience, we use the following notations and definitions established by Shi and Fuchs.[3]

- *parallel test vector (PTV)*: vector applied to all drivers in parallel at the same time.
- *sequential test vector (STV)*: vector applied to a driver in serial throughout the verification process.
- *verification pattern set (S)*: collection of all STVs. $S = \{STV_1, STV_2, \ldots, STV_m\}$. Each STV can have a different bit length, and the bit length of the longest STV is the number of PTV patterns required.
- *sequential response vector (SRV)*: response vector observed at receivers. An SRV can be a vector contributed by one or more STVs. For any SRV contributed by multiple STVs, the value in its vector is a result of certain logic operations of all contributing STVs.
- *response pattern set (S′)*: collection of all SRVs. $S' = \{SRV_1, SRV_2, \ldots, SRV_n\}$.
- *syndrome*: SRV of a connection error.
- *aliasing syndrome*: syndrome resulting from a set of erroneous nets. It is the same as a correct SRV of a net not in the set.
- *confounding syndromes*: identical syndromes that result from different sets of multiple independent errors.

PCE model

While forming port connections in HDL, we categorize two types of PCEs:

- *Floating errors*. Because unnecessary ports should be removed before verification, all remaining ports must be connected. Driver $d_i \in D$ is floating if no receiver receives its value, and a receiver is floating if no driver drives it. Any floating drivers or receivers are regarded as floating errors.
- *Connection errors*. Driver $d_i \in D$ is misconnected if it is connected to any receiver r_j such that $r_j \notin R(d_i)$. Receiver $r_i \in R$ is misconnected if it is connected to any driver d_j such that $d_j \notin D(r_i)$. That is, a port that is not connected as specified in the port description file is regarded as a connection error.

A floating error is analogous to an open fault in testing, and a connection error is analogous to a short fault or a combination of several short and open faults among ports.

Verification pattern generation

We have developed methods for generating the minimum number of patterns with DR2 capability for port connection verification. Our methods generate verification patterns for fundamental *n*-to-*n* port connections and more generalized *m*-to-*n* port connections.

Fundamental n-to-n port connections

$I_{nn}(D, R, W)$ of error-free port connections is simplified from the PCM with two extra constraints: $m = n$ and $d(w_i) = r(w_i) = 1$, for all $w_i \in W$. That is, all nets are simple nets.

Shi and Fuchs proved $\lceil \log_2(n + 2) \rceil$ test patterns to be necessary and sufficient for reaching the lowest diagnosis resolution, DR1. It is also the lower bound of the number of required patterns in interconnect testing.

Theorem 1. To diagnose all errors in $I_{nn}(D, R, W)$—that is, to reach DR2—$\lceil \log_2(n + 1) \rceil + 1$ verification patterns are necessary and sufficient.

Proof of necessity:

1. To identify each and every driver, a unique bit string (STV) is required for each and every driver. This implies that $\lceil \log_2(n) \rceil$ patterns are required.
2. To detect any receiver that receives fixed logic value 0 or 1, neither an all-0 nor an all-1 STV is allowed. A receiver that receives a fixed logic value is identified as a floating error. This implies that $\lceil \log_2(n + 2) \rceil$ patterns are required.
3. To avoid confounding syndromes in which all receivers are floating and all ports are tied together (both cases lead to an all-X value in SRVs), an extra all-0 or all-1 PTV is required. This pattern can further reduce $\lceil \log_2(n + 2) \rceil$ to $\lceil \log_2(n + 1) \rceil$ in item 2, because an all-0 PTV can avoid all-1 STVs, and an all-1 PTV can avoid all-0 STVs.

Summarizing these items, $\lceil \log_2(n + 1) \rceil + 1$ patterns are necessary.

Proof of sufficiency: After simulation, if receiver r_i receives erroneous response $SRV_{if} \neq SRV_i$, an error is detected and we can further identify errors by analyzing SRV_{if} as follows.

1. SRV_{if} is all X: Receiver r_i is floating. If not, the all-0 or all-1 PTV guarantees that 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 in which two or more drivers drive 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 bit in SRV_{if} is X: Receiver r_i is misconnected to a wrong driver. Also, we can diagnose a floating driver d_j by analyzing the verification pattern set, because we can find no contribution by the corresponding STV_j.

Neither aliasing syndromes nor confounding syndromes can occur, because a receiver would receive $SRV_{if} \neq SRV_i$ if there were any PCEs in $I_{nn}(D, R, W)$.

Generalized m-to-n port connections

$I_{mn}(D, R, W)$ of error-free port connections, which has no additional constraints, represents a more realistic and more general case of port connections. We divide verification of $I_{mn}(D, R, W)$ into two phases, each verifying a different type of net. The first phase verifies connections as n-to-n port connections without complex nets, and the second phase ensures that all complex nets are properly connected.

Phase 1. We 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, where $x = |W|$. Using the verification patterns described earlier to verify n-to-n port connections with all simple nets, we need $(\lceil \log_2(|W| + 1) \rceil = (\lceil \log_2(N_s + N_c + 1) \rceil + 1)$ 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 PCEs except those in which not all drivers on a complex net are floating. Because all drivers on a complex net are driving the same STV, the responses are not distinguishable. For example, for complex net w_1 in which $C(w_1) = \{d_1, d_2, d_3, r_1\}$, if only d_1 is floating, it cannot be detected, because 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. In this phase, we verify the connectivity of drivers on all complex nets. Note that for complex net w_i, Phase 1 ensures the connectivity of all receivers in $R(w_i)$ if they all receive the same SRV. Ensuring the connectivity of all drivers in $D(w_i)$ requires $|D(w_i)|$ patterns generated by the walking-one or walking-zero method. Therefore, to verify $I_{mn}(D, R, W)$ in Phase 2, the number of verification patterns required is $\max(|D(w_i)|)$, where $\{w_i \in W, i = 1, 2, \ldots, |W|\}$. Thus, all receivers on complex nets will receive the all-X SRV.

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 has only one bit of value 1 and all other bits of value 0; a one-cold vector has only one bit of value 0 and all other bits of value 1. For pattern set $S = \{PTV_1, PTV_2, \ldots, PTV_x\}$, which we use to verify the connectivity of drivers $\{d_1, d_2, \ldots, d_n\}$ on a complex net, we apply each PTV of value $\{v_1, v_2, \ldots, v_n\}$ 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 that driver d_i is properly connected to the complex net, we must apply a PTV to all drivers such that the bit value of v_i applied at d_i is different from all other bits. That is, the PTV must be a vector in which $v_i = 0$ and all others are 1 s (one-cold vector), or $v_i = 1$ and all others are 0 s (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 connectivity—to ensure that all drivers are indeed connected to the complex net.

Theorem 2 makes it 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, the complex net that has the most drivers determines the number of patterns required to verify all complex nets in Phase 2. The simplest way to generate such one-hot and one-cold patterns is to apply walking-one or walking-zero sequences.

In summary, to verify $I_{mn}(D, R, W)$, we need a total of $(\lceil \log_2(|W| + 1) \rceil + 1 + \max|D(w_i)|)$ to reach DR2. Because the term $(\max|D(w_i)|)$ is the number of additional patterns required for m-to-n port connections compared with n-to-n port connections, we can reduce the number of verification patterns by reducing $(\max|D(w_i)|)$. Therefore, for a complex net with ports that can be either drivers or receivers—that is, I/O ports—we can minimize patterns by configuring as many as possible I/O ports as input ports (receivers). But at least one driver must be left on a complex net for pattern application.

Table 1 shows five verification patterns derived for the example shown in Figure 2b. In Phase 1, d_2 and d_3 apply the same STV because they are on the same complex net (w_2).

Table 2 lists the expected response patterns at the receiver side after the verification patterns are applied at the driver side for the example in Figure 4. The SRVs in Phase 2 should contain all-X values.

Table 1. Verification patterns for the example in Figure 2b.

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

Response pattern resolution

By inspecting response patterns, we can easily determine whether connections are erroneous (DR1). We propose a methodology for resolving response patterns to achieve DR2 for m-to-n port connections, and it can easily be adapted to n-to-n port connections as well. Assume we have a golden response pattern set $S'_{\text{gold}} = \{SRV_1, SRV_2, \ldots, SRV_n\}$ initially, and a simulated response pattern set $S'_{\text{sim}} = \{SRV'_1, SRV'_2, \ldots, SRV'_n\}$ after simulation. Let $SRV_{i\text{-Phase1}}$ be the p-bit Phase 1 part of the receiver r_i response vector SRV_i, and let $SRV_{i\text{-Phase2}}$ be the q-bit Phase 2 part if $|D(r_i)| > 1$. Then we can write SRV_i as $\{SRV_{i\text{-Phase1}}, SRV_{i\text{-Phase2}}\}$, the concatenation of the two vectors; or as $\{SRV_i[0], SRV_i[1], \ldots, SRV_i[p + q - 1]\}$, the composition of individual bits. For each SRV_i, let $SRV_i[0]$ correspond to the all-1 or all-0 PTV used to detect floating errors. If we use $SRV'_i[0] = X$, we detect port r_i as floating. $SRV_i[1] \sim SRV_i[p - 1]$ bits are used to detect connection errors in Phase 1; the remaining bits, if any, are used to detect connection errors in Phase 2.

In addition, we use a connection matrix (CM) to represent the relationships between drivers and receivers. A CM is an $|R| \times |W|$ (that is, $n \times t$) matrix in which each element c_{xy}, where $x = 1 \sim n, y = 1 \sim t$, is a binary value indicating the connection relation-

Table 2. Expected response patterns for the example in Figure 4.

	Response pattern set (S′)	
Receiver	**Phase 1**	**Phase 2**
r_1	001	
r_2	010	XX
r_3	010	XX
r_4	011	
r_5	011	

```
1  foreach $c_{ij}$ in CM
2    $c_{ij}$ = 0
3
4  foreach $r_i$ in R
5    $r_i$.pce = 0
6    if $SRV'_i[0]$ = X
7      $r_i$.pce = 1 // $r_i$ is floating
8    else
9      Mark possible connections in CM for $r_i$ according to $SRV'_{i\text{-Phase1}}$
10     if( $SRV'_{i\text{-Phase1}}$ contains any X value )
11       $r_i$.pce = 2 // Phase 1 failed
12       Add $r_i$ into $S_x$
13     else // $SRV'_{i\text{-Phase1}}$ contains no X value
14       if( $SRV'_{i\text{-Phase1}}$ is only possibly driven by exact one driver )
15           Add $r_i$ into $S_{x'}$
16       if( $SRV'_{i\text{-Phase1}} \neq SRV_{i\text{-Phase1}}$ )
17         $r_i$.pce = 2 // Phase 1 failed
18       else
19         if( $SRV'_{i\text{-Phase2}} \neq \varnothing$ ) // Check Phase 2 response patterns
20           if( $SRV'_{i\text{-Phase2}}$ contains all-X ) // all drivers in D($r_i$)are connected
21             Add $r_i$ into $S_{x'}$ // $r_i$ is error-free
22           else
23             $r_i$.pce = 3 // Phase 2 failed
24
25 foreach $r_i$ in $S_{x'}$
26   Eliminate impossible connections in CM for each receiver in $S_x$
27
28 foreach $r_i$ in R
29   if ($r_i$.pce == 1)
30     Report $r_i$ as floating error
31   else if ($r_i$.pce == 2)
32     Report $r_i$ as Phase 1 error and print out erroneous connections
33   else if ($r_i$.pce == 3)
34     Report $r_i$ as Phase 2 error and print out absent drivers
```

Figure 3. Pseudocode for resolving response patterns.

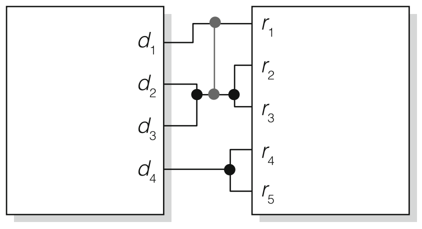
ship between receiver r_x and a group of drivers on the same net w_y. If a connection exists between receiver r_x and driver group $D(w_y)$, then $c_{xy} = 1$; otherwise, $c_{xy} = 0$.

Figure 3 shows the procedure for resolving S'_{sim}. After necessary initialization, we mark r_i as floating if its $SRV'_i[0] = X$ in line 7. In line 9, connection relationships between r_i and driver groups are updated in the CM by analyzing $SRV'_{i\text{-Phase1}}$. In lines 10 to 12, r_i is marked as a Phase 1 error if its $SRV'_{i\text{-Phase1}}$ contains any X value, and it is categorized in set S_x. In lines 14 and 15, r_i is categorized in set $S_{x'}$ if $SRV'_{i\text{-Phase1}}$ contains no X value and the value of $SRV'_{i\text{-Phase1}}$ is possibly driven only by a driver group that has exactly one driver. We can use the information of S_x and $S_{x'}$ later to eliminate impossible connections in the CM. In lines 16 and 17, r_i is marked as a Phase 1 error if its response vector is not as expected. Lines 19 to 23 analyze Phase 2 response patterns only if they have passed Phase 1.

In Phase 2, we add r_i to $S_{x'}$ if it is error free; otherwise, we mark it as a Phase 2 error. In lines 25 and 26, we use the connection information in $S_{x'}$ to identify impossible connections S_x. That is, the drivers involved with the receivers in $S_{x'}$ are not possibly involved with those in S_x; otherwise, the receivers in $S_{x'}$ would also receive an SRV with some X values. (This is trivial and not proved here.) Finally, in lines 28 to 34, we print out erroneous receivers.

Figure 4 shows an example of erroneous port connections for Figure 2b. We use this example to demonstrate the procedure of pattern resolution. Table 3 shows the simulated response patterns observed at the receivers after we apply the verification patterns shown in Table 1.

Figure 5 shows golden and erroneous CMs. All the receivers are free-floating, since all $SRV'_i[0] = 0$. We analyze $SRV'_i[1]$ and $SRV'_i[2]$ to mark possible connections in the CM. For example, $\{SRV'_1[1], SRV'_1[2]\} = \{XX\}$, and thus r_1 is possibly connected to driver groups $D(w_1)$, $D(w_2)$, and $D(w_3)$, which



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

Figure 4. Erroneous port connections, and the corresponding erroneous HDL code shown in bold.

apply patterns of 01, 10, and 11, respectively. In this example, r_1, r_2, and r_3 are in S_x, and r_4 and r_5 are in $S_{x'}$. After inspecting S_x and $S_{x'}$, we can remove the connections between $D(w_3)$ and r_1, r_2, and r_3 (Figure 5c), to reduce the number of error candidates for inspection. As a result, we mark r_1, r_2, and r_3 as erroneous and r_4 and r_5 as error free. In addition, we can easily identify misconnected receivers r_1, r_2, and r_3 by comparing the CM in Figure 5c with the golden one in Figure 5a.

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

	Response pattern set (S'_{sim})	
Receiver	**Phase 1 {$SRV'_i[0]$, $SRV'_i[1]$, $SRV'_i[2]$}**	**Phase 2 {$SRV'_i[3]$, $SRV'_i[4]$}**
$r_1(SRV'_1)$	0XX	
$r_2(SRV'_2)$	0XX	XX
$r_3(SRV'_3)$	0XX	XX
$r_4(SRV'_4)$	011	
$r_5(SRV'_5)$	011	

$$\begin{array}{cccc} & D(w_1) & D(w_2) & D(w_3) \\ r_1 & 1 & 0 & 0 \\ r_2 & 0 & 1 & 0 \\ r_3 & 0 & 1 & 0 \\ r_4 & 0 & 0 & 1 \\ r_5 & 0 & 0 & 1 \end{array}$$

(a)

$$\begin{array}{cccc} & D(w_1) & D(w_2) & D(w_3) \\ r_1 & 1 & 1 & 1 \\ r_2 & 1 & 1 & 1 \\ r_3 & 1 & 1 & 1 \\ r_4 & 0 & 0 & 1 \\ r_5 & 0 & 0 & 1 \end{array}$$

(b)

$$\begin{array}{cccc} & D(w_1) & D(w_2) & D(w_3) \\ r_1 & 1 & \mathbf{1} & 0 \\ r_2 & \mathbf{1} & 1 & 0 \\ r_3 & \mathbf{1} & 1 & 0 \\ r_4 & 0 & 0 & 1 \\ r_5 & 0 & 0 & 1 \end{array}$$

(c)

Figure 5. Golden and erroneous connection matrices (CMs): golden (a); erroneous (b); and erroneous with the connections between driver group $D(w_3)$ and receivers r_1, r_2, and r_3 removed (c), with bold items indicating the differences between (a) and (c).

Port connection verification flow

The verification flow we propose requires two port connection formats. One format can be a description written in any HDL. The other format describes the connections of all ports according to the specification and can be as simple as a two-column table—one column listing the drivers, and the other listing the corresponding receivers.

Figure 6 shows our verification flow. Once designers write the design specification and refine all interfaces to pin accuracy, they integrate necessary IP blocks and code in HDL code. Meanwhile, the system maintains a file for port connection descriptions. Processes in the shaded area in Figure 6 are automatic. Verification and expected response patterns are generated from the port connection description file. A testbench automatically generated from the HDL design will be used for simulation later. All modules in the testbench are stub models containing only the port interface information extracted from the HDL design. An additional process written in HDL applies verification patterns. Response patterns are resolved by another process written in HDL, or are dumped into a file and resolved by other programs offline.

After simulation, a pass signal is asserted if port connections are correct. Otherwise, a fail signal is asserted, and all PCEs are reported. Designers can verify the port connections on the basis of the reported information. They should use the verification flow again whenever the port connections are modified.

Although manual verification of port connections is not a complex task, it is extremely time-consuming, tedious, and error-prone when there are many ports.

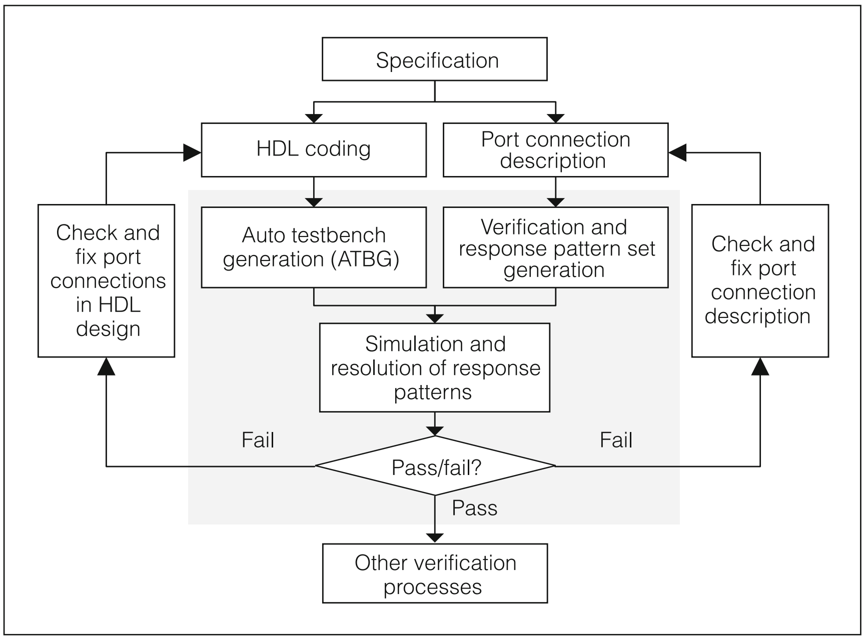

Specification
HDL coding
Port connection description
Check and fix port connections in HDL design
Auto testbench generation (ATBG)
Verification and response pattern set generation
Check and fix port connection description
Simulation and resolution of response patterns
Fail
Fail
Pass/fail?
Pass
Other verification processes

Figure 6. Port connection verification flow.

This is especially true now that SoC designs are more complex and contain more IP blocks with tens of thousands of ports. Also, design configurations sometimes change to accommodate different performance and cost requirements during the design exploration process. This augmentation, exchange, or removal of IP blocks leads to repeated modification of port connections. Thus, port connection errors can occur repeatedly, and must be verified whenever port connections are modified. Designers can start the verification flow we propose as soon as all the IP interfaces in the system are refined to pin accuracy, and they can detect and diagnose multiple errors. With our methodology, designers can save considerable time compared with verifying port connections manually. ■

■ References

1. W. Feng, F.J. Meyer, and F. Lombardi, "Two-Step Algorithms for Maximal Diagnosis of Wiring Interconnects," *Proc. 29th Ann. Int'l Symp. Fault-Tolerant Computing* (FTCS 99), IEEE CS Press, 1999, pp. 130-137.
2. J.-C. Lien and M.A. Breuer, "Maximal Diagnosis for Wiring Networks," *Proc. Int'l Test Conf.* (ITC 91), IEEE Press, 1991, pp. 96-105.
3. W. Shi and W.K. Fuchs, "Optimal Interconnect Diagnosis of Wiring Networks," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 3, no. 3, Sept. 1995, pp. 430-436.
4. W.-T. Cheng, J.L. Lewandowski, and E. Wu, "Optimal Diagnostic Methods for Wiring Interconnects," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 9, Sept. 1992, pp. 1161-1166.
5. D. McBean and W.R. Moore, "Testing Interconnects: A Pin Adjacency Approach," *Proc. 3rd European Test Conf.* (ETC 93), IEEE Press, 1993, pp. 484-490.
6. J. Zhao, F.J. Meyer, and F. Lombardi, "Analyzing and Diagnosing Interconnect Faults in Bus-Structured Systems," *IEEE Design & Test*, vol. 19, no. 1, Jan./Feb. 2002, pp. 54-64.
7. X.T. Chen, F.J. Meyer, and F. Lombardi, "On the Fault Coverage of Interconnect Diagnosis," *Proc. 15th VLSI Test Symp.* (VTS 97), IEEE CS Press, 1997, pp. 101-107.
8. Y. Kim, H.-D. Kim, and S. Kang, "A New Maximal Diagnosis Algorithm for Interconnect Test," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, May 2004, pp. 532-537.
9. N. Jarwala and C.W. Yau, "A New Framework for Analyzing Test Generation and Diagnosis Algorithms for Wiring Interconnect," *Proc. Int'l Test Conf.* (ITC 89), IEEE CS Press, 1989, pp. 63-70.
10. A. Hassan, J. Rajski, and V.K. Agarwal, "Testing and Diagnosis of Interconnects Using Boundary Scan Architecture," *Proc. Int'l Test Conf.* (ITC 88), IEEE CS Press, 1988, pp. 126-137.
11. J.C. Chan, "Boundary Walking Test: An Accelerated Scan Method for Greater System Reliability," *IEEE Trans. Reliability*, vol. 41, no. 4, Dec. 1992, pp. 496-503.
12. C.-Y. Wang, S.-W. Tung, and J.-Y. Jou, "On Automatic Verification Pattern Generation for SOC with Port Order Fault Model," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 4, Apr. 2002, pp. 466-479.

Geeng-Wei Lee is pursuing a PhD in the Department of Electronics Engineering at National Chiao Tung University, Hsinchu, Taiwan. His research interests include design verification, VLSI testing, and high-level design methodologies. He has an MS in electrical engineering from National Cheng Kung University.

Juinn-Dar Huang is an assistant professor in the Department of Electronics Engineering of National Chiao Tung University, Hsinchu, Taiwan. His research interests include computer-aided design and verification, embedded-processor design, and system-level design methodology. He has a PhD in electronics engineering from National Chiao Tung University.

Chun-Yao Wang is an assistant professor in the Computer Science Department at National Tsing Hua University, Taiwan. His research interests include logic synthesis, design verification, and VLSI testing. He has a PhD in electronics engineering from National Chiao Tung University.

Jing-Yang Jou is the vice chancellor of the University System of Taiwan (consisting of National Central, National Chiao Tung, National Tsing Hua, and National Yang Ming universities). His research interests include logic and system synthesis, design verification, CAD for low power, and networks on chips. He has a PhD in computer science from the University of Illinois at Urbana-Champaign.

■ Direct questions and comments about this article to Geeng-Wei Lee, Dept. of Electronics Engineering, National Chiao Tung University, 1001 Ta-Hsueh Road, Hsinchu, Taiwan 300 ROC; gwlee@eda.ee.nctu.edu.tw.