Short Paper_____

Using Stack Reconstruction on RTL Orthogonal Scan Chain Design[*]

CHIA-CHUN TSAI[+], HANN-CHENG HUANG, TRONG-YEN LEE,
WEN-TA LEE AND JAN-OU WU
[+]*Department of Computer Science and Information Engieering*
Nanhua University
Chiayi, 622 Taiwan
E-mail: chun@mail.nhu.edu.tw
Graduate Institute of Computer and Communication Engineering
National Taipei University of Technology
Taipei, 106 Taiwan

In this paper, we propose an orthogonal scan chain embedded into the RTL design described by Verilog. We first construct the data path graph from the embedded scan chains and then find all the orthogonal scan paths with the minimum weighted cost. These paths share with original data paths as possible. Finally, we create a stack form to reconstruct all the orthogonal scan paths to manage the I/Os and reduce the length and width of stack form as well as decrease the overheads of area and timing. Experimental results show our RTL orthogonal scan chain approach can save up to 13.8% and 5.1% in area overhead than that of the gate-level scan and functional order RTL scan, respectively.

Keywords: RTL scan, orthogonal scan chain, area overhead, timing overhead, fault coverage

1. INTRODUCTION

Scan chain is widely used to improve the testability of circuits, especially for sequential logics. The scan chain is inserted into circuits such that allow us to assign test patterns to the inputs of circuits (*i.e.*, maximum controllability) and then observes their output responses (*i.e.*, maximum observability) to know where possible faults are. But, the effects of area and timing overheads should be associated with inserted scan chains for the evaluation of testability.

Traditionally, scan chains are directly inserted into a gate-level circuit. Reddy and Dandapani [1] first proposed a boundary scan that used standard flip-flops for the scan path design. More multiplexers were inserted into the gate-level circuits for the testability that caused higher fault coverage but larger area and timing overheads. The approach is a

_____

typical DFT (design for testability) technique. Lin *et al.* [2, 3] established scan paths by adding constraints at the primary inputs and inserting test points into the *free-scan* flip-flops through the existed combinational logics to reduce the area overhead. But, the identification of scan paths and the insertion of test points were shared for testability that would lead to very timing overhead, and they will possibly make iterative redesign for matching the user constraints and optimizing the circuits that would cause area overhead.

To overcome above problems, scan chains are considered to first make insertion at RTL (register-transistor logic) levels before synthesizing to gate-level circuits. Thus, scan chain path can share existed original data paths that should reduce area and timing overheads. This approach can take advantages of global design optimization performed by synthesis tools and satisfies the constraints defined by users.

Asaka *et al.* proposed H-SCAN+ [4] and H-SCAN [5] started from a structural RTL to insert scan chains between the existed paths of registers and multiplexers for reducing area overhead. It may be concerned that some other functional paths are not exploited. Aktouf *et al*. [6] proposed a random order RTL scan chain. The scan chain is stitched the registers together randomly at the behavioral level. But, it does not exploit the original data paths for reducing the area overhead in advance.

Norwood and McCluskey [7, 8] investigated an orthogonal scan chain associated with high-level synthesis to apply multiple tests. During the allocation and binding of hardware resources, the data paths shared with each other may cause extra area and timing overheads. Huang *et al.* [9] proposed a functional order RTL scan chain. The approach first divides a module into several processes existed in sequence on an RTL code and then finds the scan path for each process with the minimum weighted cost and the consideration of sharing functional logics. The results show that the approach is better than the gate level and random order RTL scans in area overhead.

In this paper, we proposed a low-overhead scan design by inserting orthogonal scan chains associated with the functional order RTL level. We find all the orthogonal scan data paths with the minimum weighted cost and try to share with the consideration of existed data paths as possible. Experimental results show that our proposed method can achieve lower area and timing overheads to compare with the traditional gate-level scan, random order RTL scan, and functional order RTL scan.

The remainder is organized as follows. Section 2 introduces the system overview. The algorithm of the proposed orthogonal RTL scan chain is illustrated in details in section 3. Section 4 shows the experimental results. Finally, a conclusion remark is presented in section 5.

2. SYSTEM OVERVIEW

Fig. 1 shows the system overview of our orthogonal RTL scan chain. The input of the system is an RTL circuit which is described in Verilog HDL. Since a circuit with the Verilog HDL code can be represented hierarchically, an RTL circuit consists of at least a module and a module may include other sub-modules and a sub-module can contain the other sub-modules. The hierarchy is just for our major reference to the insertion ordering planning of scan chains.

We first scan the circuit from its RTL code and then extract all the hierarchical
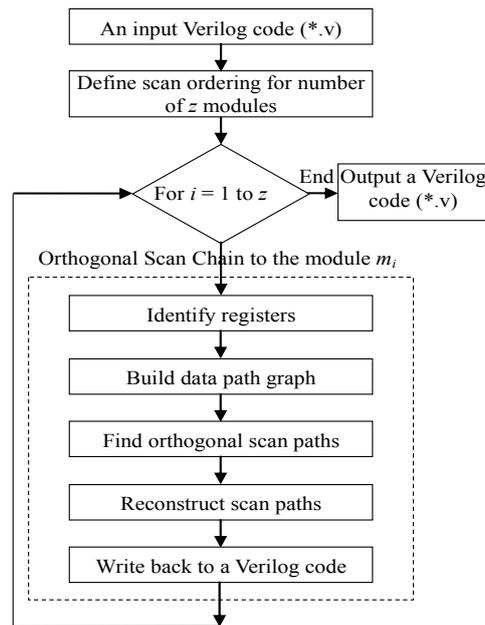
Fig. 1. The system overview of orthogonal RTL scan chain design.

modules of z, m_1, m_2, …, m_z, and identify their relationship among modules. According to the module relationship, we can build the directed module graph. The directed module graph displays the natural scan ordering among modules. This relation derives us to plan accordingly the module ordering of scan chain insertion. Obviously, our scan chain is the bottom-up design.

Second, we insert scan chains into each module individually. For each module m_i, $1 \leq i \leq z$, we identify all the inputs, registers, and outputs from its RTL code and then build a data path graph (DPG) according to their relations. From the DPG, we can find out all the data paths with the minimum weighted cost sequentially. Since some of scan paths may not directly connect to inputs or outputs and some of registers may not match the width of scan paths, we create a stack form to reconstruct the scan paths and to minimize the length of scan paths. Finally, we modify the Verilog code to the module that has been inserted orthogonal RTL scan chains for the evaluation of testability.

The above procedure is repeated until all the modules are completed with the orthogonal RTL scan chain insertion.

3. ORTHOGONAL RTL SCAN CHAIN

The following subsections for the orthogonal RTL scan chain are described in details.

3.1 Identify Registers

In Verilog code, the *always* statement is a structural procedure that executes an activity flow repeatedly at the behavioral level. The *always* block defined in a module as-

sociates with the edge trigger, *posedge* or *negedge*, that can be mapped to a sequential circuit which consists of multiplexers and flip-flops. But, the *always* block without any edge trigger will be mapped to multiplexers in general. Thus, we can identify all the *always* blocks with edge triggers in a module for the search of data paths.

For example, a Verilog code is given in Fig. 2. There are three 8-bit registers, A, B, and C, that have two *always* blocks in the module of MOVE, but just the 8-bit register A in the *always* block with the positive edge trigger of *posedge* can be mapped to a sequential circuit.

```
module MOVE(clock, reset, addr);
input clock, reset, addr;
reg [7:0] A, B, C;
always@(posedge clock) begin      // positive edge trigger
   if (addr)
      A <= B − 1;
   else
      A <= C;
end
always@(A or B) begin             // no edge trigger
   if (addr)
      B <= C + 1;
   else
      C <= A;
end
endmodule
```

Fig. 2. An RTL example of identifying registers from a module.

3.2 Build Data Path Graph

An *always* block with or without the edge trigger may have three kinds of data paths. First, delay data path is used to move a data stored in one register *i* directly to another register *j* but just spend a little delay, such as the register B transfers its data to the register A, *i.e.*, A <= B in a Verilog code. Second, conditional data path, such as the statement of *if … else* or *case*, is used to transfer the data stored in one register *i* or *j* to another register *k* determined by the selection of multiplexers. Third, operational data path, such as all the statements of arithmetic and logic operations, is used to deliver the output data operated from two input registers *i* and *j* of a functional block to another register *k*. We are expecting to know all the sequential data paths within a module for the consideration of scan chain insertion for testability. The data path graph (DPG) can help us for the resolution.

The DPG of a module is defined a directed graph $G = (V, E)$, where the set of vertices $V = \{v_1, v_2, …, v_n\}$ represent the inputs, intermediate registers, and outputs, and a set of directed edges $E = \{e_1, e_2, …, e_m\}$ represent the data paths between vertices.

If v_i has a data path toward v_j via functional logics or bypass, then a directed edge from v_i to v_j will be constructed in the DPG. With each directed edge, we associate a weight that defines the cost which is used to guide its data path as the scan path possible

for reducing area and timing overheads. The weight cost is determined according to the following rules.

Rule 1: If there exists a delay data path from the register i to the register j, then the edge weight of their corresponding nodes v_i to v_j in the DPG is set to be 0. That is, $e(v_i, v_j) = 0$. The setting is due to the scan path can always share with the original functional data path between vertices v_i and v_j that will cause no extra area added.

Fig. 3 shows the example of RTL code that has a delay data path from the byte-level register A to B and their synthesized logics before and after scan chain insertions. From Fig. 3 (b), there is no any area overhead as inserting a scan path because the scan path shares with the original functional path from the register A to B.
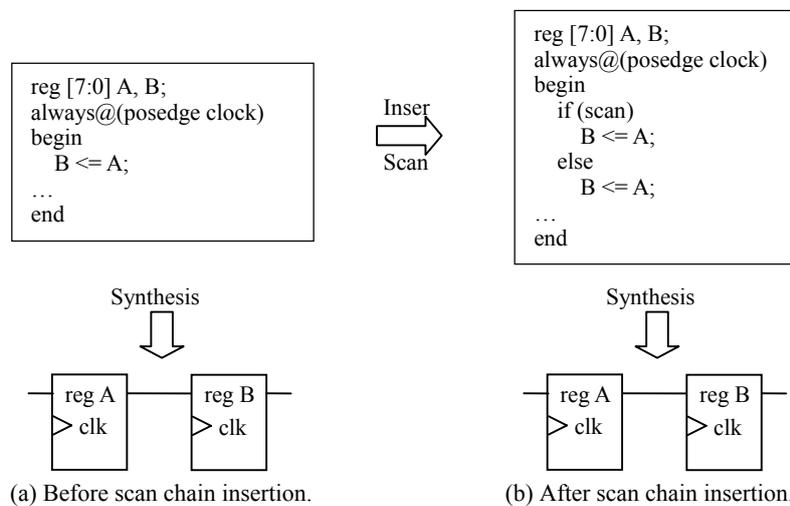

Fig. 3. An RTL example of the weight 0 for Rule 1.

Rule 2: If there exists a conditional data path from the register i or j to the register k, then the edge weights of their corresponding nodes v_i to v_k or v_j to v_k in the DPG are set to the data width of the register i or j. That are, $e(v_i, v_k) = N_1$ and $e(v_j, v_k) = N_2$, where N_1 and N_2 are the number of bits that is the data width of the registers i and j, respectively. The setting is considered that the number of control gates are required for the selection from the register i or j for the account of area overhead.

Fig. 4 shows the example of RTL code that have a conditional data path from the register B or C to A, and their synthesized logics before and after scan chain insertions. From Fig. 4 (b), the number of N new two-input OR gates are created for selecting the register B or C from the multiplexers to the register A by the control signal *scan_en* after the scan chain insertion. This can accommodate the scan paths usage through the multiplexers for the account of area overhead.
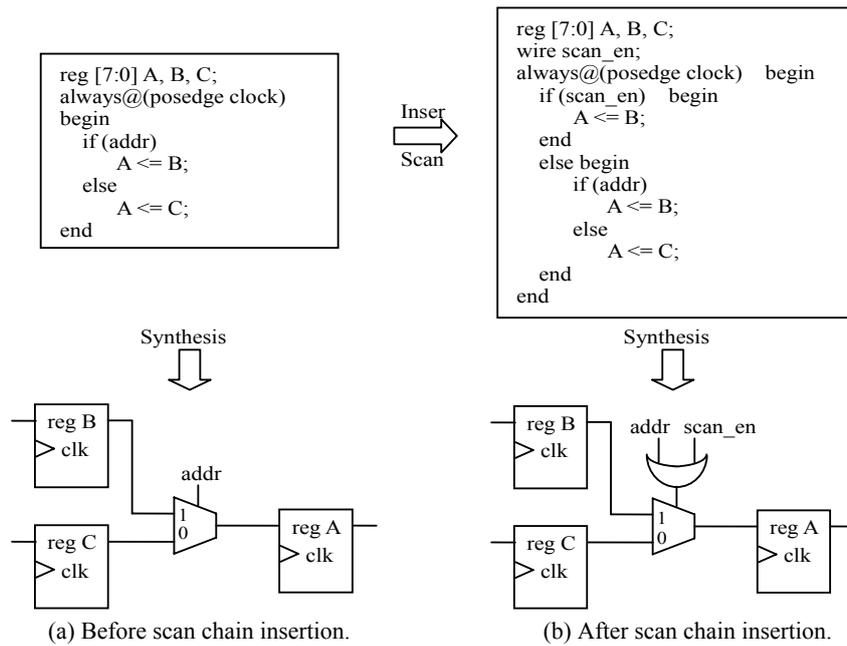
Fig. 4. An RTL example of the weight *N* for Rule 2.

(a) Before scan chain insertion.   (b) After scan chain insertion.
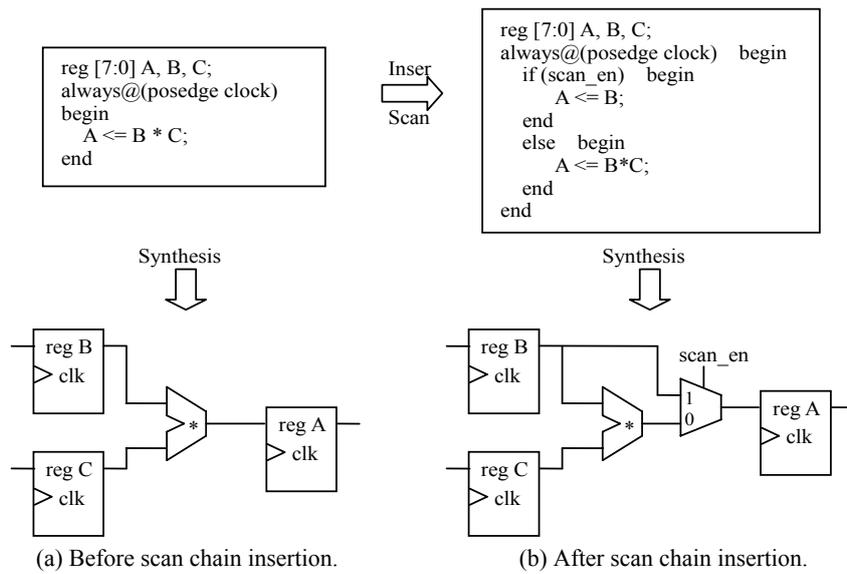
Fig. 5. An RTL example of the weight 3*N* for Rule 3.

(a) Before scan chain insertion.   (b) After scan chain insertion.

Rule 3: If there exists an operational data path from two input registers *i* and *j* to the register *k* by making arithmetic or logic operations, then the edge weights of their corresponding nodes v_i to v_k or v_j to v_k in the DPG are set to the triple data width of the register

i or *j*. That are, $e(v_i, v_k) = 3N_1$ and $e(v_j, v_k) = 3N_2$, where N_1 and N_2 are the data width of the register *i* and *j*, respectively. The setting is taken that the number of multiplexers are required for the selection from their operational results or the register *i* or *j* and a multiplexer is assumed three times of a basic gate for the account of area overhead

Fig. 5 shows the example of RTL code that have an operational data path from the multiplication of A <= B * C and their synthesized logics before and after scan chain insertion. From Fig. 5 (b), a new multi-input multiplexer is created for the selection from the multiply result of B * C or the register B to the register A by the control signal *scan_ en* after the scan chain insertion.

According to the above rules, we can insert orthogonal scan chains into the modules of the RTL code in a circuit and build a byte-level data path graph (DPG) for the circuit. Fig. 6 shows the DPG of the example of an 8-bit multiplier. From the figure, the edge weights of *e*(in1, temp1) and *e*(in2, temp2) are 0 due to they have the delay data paths of the statements temp1 <= in1 and temp2 <= in2, the edge weights of *e*(temp1, temp4) and *e*(temp1, temp4) are the data width of 8 because of their conditional data paths, and the edge weights of *e*(temp1, temp3), *e*(temp1, temp3), *e*(temp3, out1), *e*(temp4, out1) are the triple data width of 24 because of their operational data paths.

```
module MUL(in1, in2, out1, reset, clock, addr);
input clock, reset, addr;
input [7:0] in1, in2;
output [15:0] out1;
reg [7:0] temp1, temp2, temp3, temp4;
always@(posedge clock)    begin
   temp1 <= in1; temp2 <= in2;
   if (addr)    begin
        temp3 <= temp1 – 10; temp4 <= temp2;
   end
   else    begin
        temp3 <= temp2 + 10; temp4 <= temp1;
   end
   out1 <= temp3 * temp4;
end
endmodule
```

(a) Verilog code.   (b) Data path graph.
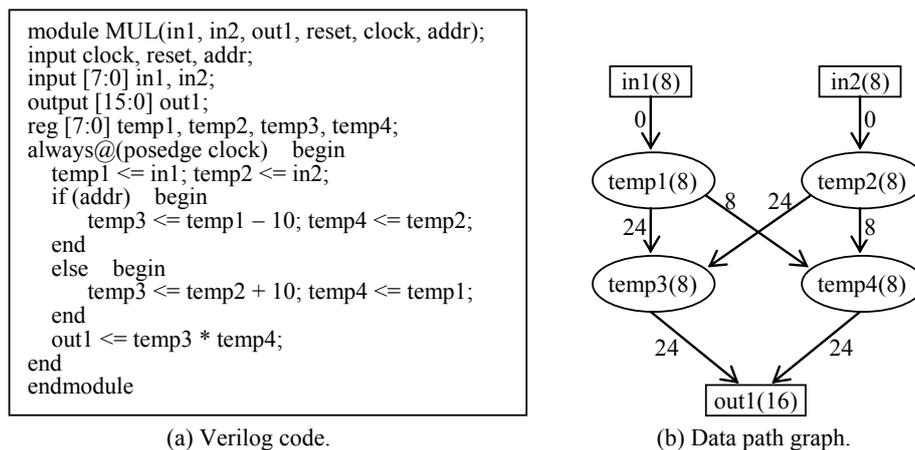
Fig. 6. The DPG of an 8-bit multiplier.

3.3 Find All Scan Paths

The DPG of a module is obtained from subsection 3.2, we adopt a greedy method to identify all the possible scan paths (SPs) with low weights from the DPG. A low-weight scan path may contribute lower area overhead for testability.

We first start from each node that has no any predecessor in a DPG to find a scan path with the low weight to an ended node that has no any pointer to other nodes. Then select the scan path with the minimum weight as well as maximum number nodes from the above finding low-weight scan paths as our first scan path. Then remove all the edges

that their path nodes point to other nodes or are pointed from other nodes from the DPG. The above procedure is repeated for next scan paths until the DPG is empty.

The algorithm is shown in the following. Obviously, the time complexity is $O(n^2)$ because two loops of *While* and *For* are nested, where n is the number of nodes in a DPG.

Algorithm Scan_path_search (DPG)
{ SP = {};   /* scan path is empty */
   While (DPG is not empty)
   { Find all the nodes X from the DPG that have no any predecessor;
      For (i = 1 to $|X|$)
      { Find a scan path with the low weight including maximum number nodes from v_i to
        a node in the DPG that has no any point to other nodes
      }
      Select the scan path SP_j with the minimum weight from the number of $|X|$ low-weight scan paths;
      Remove all the edges from the DPG that the nodes of the path SP_j point to other nodes or are pointed from other nodes;
      $SP = SP + \{SP_j\}$;
   }
}

For example, the DPG as shown in Fig. 6 (b), we can get the first scan path SP1, in1 $\rightarrow$ temp1 $\rightarrow$ temp4 $\rightarrow$ out1, with the minimum weight of 32 (*i.e.*, 0 + 8 + 24) and maximum nodes of 4 and then get the second scan path SP2, in2 $\rightarrow$ temp2 $\rightarrow$ temp3. Fig. 7 shows two final scan paths SP1 and SP2.
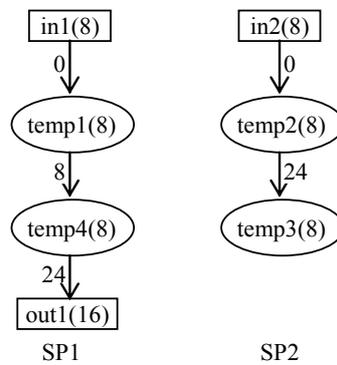
Fig. 7. Two scan paths found from the DPG shown in Fig. 6 (b) of an 8-bit multiplier.

3.4 Reconstruct Orthogonal Scan Chains

The scan paths for a module have been obtained in subsection 3.3, but it is noted that some of scan paths may not start from an input node or end to an output node and the data widths of all the registered nodes of a path are not always equal with each other. These fragmentary scan paths are inconvenient for testability.

For instance, two scan paths *SP*1 and *SP*2 as shown in Fig. 7, the *SP*1 starts from the input in1 and ends to the output out1, but has two different data widths, 8 for the in1 and internal registers temp1 and temp4 and 16 for the output out1. The scan path *SP*2 has a consistent data width, but stops at the internal register temp3 that is not for the output observables. Thus, we need to reconstruct the scan paths such that all the refined scan paths can be tested by assigning test patterns to the inputs and then observe the outputs.

We first construct the scan path *SP*1 and create a stack form to restore the 8-bit input in1, two intermediate 8-bit registers temp1 and temp4, and the 16-bit output out1. Then we continue to construct the next scan path *SP*2 into the stack form, the 8-bit input in2 is pushed on the above of the 8-bit input in1, the intermediate 8-bit registers temp2 and temp3 are pushed on the above of the 8-bit registers temp1 and temp4, respectively. Finally, we get the scan chain from the stack form as shown in Fig. 8, that is {in1, in2} → {temp1, temp2} → {temp4, temp3} → {out1}.
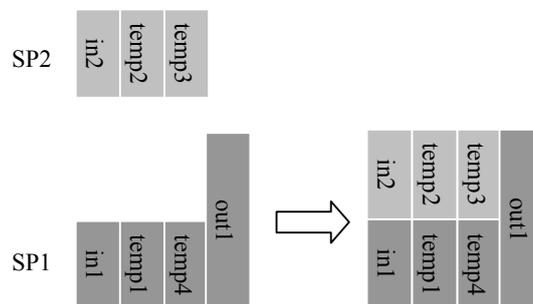

Fig. 8. An example of reconstructing two scan paths shown in Fig. 7 by a stack form.

From the above idea of the stack form usage for reconstructing all the scan paths, the stack form consists of three parts, the inputs, intermediate registers, and outputs. We can image the stack form is a channel that is used to transfer data from the input to the output for the maximum width as possible.

With the idea, we create a stack form of $N \times M$ that is used to reconstruct given data paths into final scan chains with lower area and timing overheads, where N and M are the width and length of the stack form, respectively. The length M can be various with the maximum length of scan chains. That is the sum of the number of columns in the stack form, such as $M = 4$ for Fig. 8. The width N has a threshold value that depends on the sum Nin of data widths of inputs, the maximum sum Nreg of data widths of intermediate registers each column, and the sum Nout of data widths of outputs. If Nin < Nout or Nout < Nin, we let N be the smallest to limit the largest for smoothing data streams in scan chains. If Nreg > min(Nin, Nout), we let N be the Nreg to allow to push more intermediate registers into a registered column. Thus we have a formula for the calculation of the width N as following.

$$N = \max(\min(N\text{in}, N\text{out}), N\text{reg}).   \tag{1}$$

For instance, the width $N = 16$ for Fig. 8. For a DPG, if each node (an input, a reg-

ister, or an output) is stacked at a column, the length M is defined as the total number of nodes and the maximum width N is defined as the sum of data widths of the maximum width of nodes in the DPG.

Since we want to keep orthogonal scan paths associated with their low-weight originality as possible, our approach to the reconstruction order for all the scan paths by the experimental priority of the scan path with an input but no any output, the path without any input and output, the path with both input and output, and the path with an output but no any input. If the node of a scan path is an input node, then push the input node on the input located on the first column of the stack form. If the node of a scan path is an internal node, then push the register node on the first columns that are stacked for intermediate registers as tight as possible. If the node of a scan path is an output node, then push the output node on the output located on the last column of the stack form.

With the stack form for making the reconstruction of all the scan paths, there are two benefits. One is to exploit I/Os to generate maximum scan chains and another is to reduce the lengths of scan chains. But, the final scan chains may change some of low-weight orthogonal scan paths and increase the area overhead. However, their effects in our proposed are limited for most of cases compared with other approaches.

The algorithm for reconstructing the integrated scan chains is presented as following. Obviously, the time complexity is $O(n^2)$ because two *For* loops are nested, where n is the number of nodes in a DPG.

Algorithm Reconstruct_scan_chain (DPG, SP)
{ Create a stack form of $N \times M$ that is used for the *SP* reconstruction, where the length M
     = total number of nodes of the DPG and the width N is defined as Eq. (1);
  Sort the *SP* with the priority order of the path with an input but no any output, the path
  without any input and output, the path with both input and output, and the path with an
  output but no any input;
  For (i = 1 to $|SP|$)
    { For (j = 1 to $|SP_i|$)
      { Case 1: Vj = input node;
              Push Vj on the input located on the 1st column of the stack form;
        Case 2: Vj = internal node;
              Push Vj on the first columns that are stacked for intermediate registers
              as tight as possible;
        Case 3: Vj = output node;
              Push Vj on the output located on the last column of the stack form;
      }
    }
  Try to reduce the length of scan chains.
}

4. EXPERIMENTAL RESULTS

We conduct experiments on our approach with the benchmarks of ITC'99 [2]. We first translate the examples of B01 to B11 edited by VHDL codes to be Verilog codes.

The example B08 is not presented here because we cannot translate it due to amount of RAMs. Then we make the insertion of proposed orthogonal scan chains to the Verilog RTL codes. The Synopsys Design Compiler is used to synthesize the Verilog codes optimally to be logic circuits represented with the gate-level net list and the Synopsys Design Analyzer is employed to calculate the consumed area units from the gate-level net list for the evaluation of area overhead. Finally, the Synopsys TetraMax is used to generate test patterns for the evaluations of fault coverage and test cycles.

Tables 1 and 2 show the comparisons with other scan chains in terms of area overhead and fault coverage, respectively, for ITC'99 benchmarks. The traditional gate-level results are obtained from that the Verilog RTL codes are first synthesized to be the gate-level logics and then the Synopsys Test Compiler is applied to insert scan chains with multiple standard multiplexers. For the results of functional-order RTL scan, we first insert scan chains manually into the VHDL codes according to the implementation steps referred in Fig. 9 of Huang *et al*. [9] and then translate them to be Verilog codes.

Table 1. Comparison with other scan chains in term of area overhead (A.O.).

ITC'99	#FF	Before Scan	Gate level [1]		Random order RTL [6, 9]		Functional order RTL [9]		Ours	
		Area	Area	A.O.	Area	A.O.	Area	A.O.	Area	A.O.
B01	5	80.1	92.2	15.1%	102.3	27.7%	87.8	9.6%	89.2	11.3%
B02	4	52.1	64.1	23.0%	76.0	45.9%	73.8	41.7%	63.7	22.3%
B03	30	310.0	390.1	25.8%	394.1	27.1%	354.5	14.4%	320.0	3.2%
B04	66	640.3	878.9	37.3%	708.7	10.6%	680.4	6.3%	685.7	7.1%
B05	34	692.0	794.9	14.9%	634.4	-8.3%	629.7	-9.0%	762.0	10.1%
B06	9	112.5	132.8	18.1%	174.7	55.3%	152.1	35.3%	121.5	8.0%
B07	49	598.9	741.2	23.8%	679.4	13.4%	608.4	1.6%	658.4	9.9%
B09	28	276.7	370.4	33.9%	344.7	24.5%	329.9	19.2%	296.7	7.2%
B10	17	240.0	279.3	16.4%	258.2	7.5%	257.1	7.1%	247.0	3.0%
B11	31	569.5	663.6	16.5%	699.1	22.7%	637.0	11.9%	596.5	4.7%
Average	−	−	−	22.5%	−	22.6%	−	13.8%	−	8.7%

Note: Area is the sum of units.

From Table 1, it is noted that the results in area overhead of random-order RTL scan are just translated from the data extracted the 5th column in Table 1 of Huang *et al*. [9] by a constant scale factor. For instance, the total areas of B01 before and after the random-order scan in Table 1 of [9] are 551.53 and 704.35 respectively, it's scale constant equals 1.277 (*i.e.* 704.35/551.53) and the total area of B01 in our Table 1 is thus converted to be 102.3 (*i.e.* 80.1 × 1.277). The area overheads for most of all the examples adopted our approach in the table are obviously lower than that of other scan chains. Examples B05 and B07 have slightly increment in area overhead due to their memory structures. In summary, our area overhead on average can save up to 13.8% (22.5.0% − 8.7%), 13.9% (22.6% − 8.7%), and 5.1% (13.8% − 8.7%) than that of the gate level, random order, and functional order scan chains, respectively. But the comparison with the random-order scan is just as the reference because of the very different running environments.

Table 2. Comparison with other scan chains in term of fault coverage.

ITC'99	#FF	Before scan	Gate level [1]	Random order RTL [6, 9]	Functional order RTL [9]	Ours
B01	5	68.43%	100%	99.66%	100%	100%
B02	4	61.54%	100%	96.70%	100%	100%
B03	30	72.45%	100%	99.76%	100%	100%
B04	66	85.84%	100%	98.36%	99.5%	93.7%
B05	34	45.03%	100%	96.89%	94.24%	99.27%
B06	9	90.82%	100%	98.51%	100%	100%
B07	49	64.77%	99.6%	94.75%	99%	94.7%
B09	28	86.29%	100%	99.89%	100%	97.5%
B10	17	86.09%	100%	99.55%	100%	96.1%
B11	31	74.76%	99.95%	96.67%	98.7%	98.5%
Average	–	73.60%	99.96%	98.07%	99.14%	97.98%

Table 3. Comparison with others in terms of scan chains and clock cycles of one input test pattern.

ITC'99	#FF	#scan chain				#cycles of one input test pattern			
		Gate level [1]	Random order [6]	Functional order [9]	Ours	Gate level [1]	Random order [6]	Functional order [9]	Ours
B01	5	1	1	1	2	5	5	5	4
B02	4	1	1	1	1	4	4	4	4
B03	30	1	1	1	4	30	30	30	8
B04	66	1	1	8	8	66	66	9	9
B05	34	1	1	1	1	34	34	34	34
B06	9	1	1	1	2	9	9	9	5
B07	49	1	1	1	1	49	49	49	49
B09	28	1	1	1	1	28	28	28	28
B10	17	1	1	1	6	17	17	17	3
B11	31	1	1	6	6	31	31	6	6
Average	–	1	1	2.2	3.2	27.3	27.3	19.1	15

From Table 2, the evidences in fault coverage of random-order scan are directly referred from the 8th column in Table 1 of Huang *et al*. [9]. Our fault coverage loss is just 1% on average than others, but it closes to the reasonable requirement of 98%. Examples B04, B09, and B10 have lower fault coverage because some scan paths with clustering registers are slightly changed during the stack reconstruction.

We use one test pattern to evaluate the clock cycles spent through the scan chains of an RTL code. Table 3 shows the comparison with others in terms of number of scan chains and clock cycles to one input test pattern. The cycles in test time of random-order scan are the same that of the gate level due to the same test procedure. From the table, the

number of scan chains insertion adopted by ours are always more than others and the multiple scan chains usually get shorter test time.

Two larger circuits, 1-D DWT (discrete wavelet transform) filter [10] and 2-D CORDIC (coordinate rotation digital computer) [11], are tested in advance for verifying our approach. Table 4 presents the evaluation and shows that our results are always encouraged in terms of area overhead, number of test patterns, maximum scan length, and number of scan chains. The gate-level results are obtained with the same running environment of ours. The reduced limitation in area overhead for 2-D CORDIC is due to the circuit characteristics that each element of most memories includes at least one operand unit.

Table 4. The results of two larger examples with our orthogonal scan.

	1-D DWT [10]			2-D CORDIC [11]		
	Before scan	Gate level [1]	Ours	Before scan	Gate level [1]	Ours
Area (#unit)	2145.2	2584.5	2255.8	8666.4	10878.2	10218.1
Area overhead (%)	–	20.5%	5.2%	–	25.5%	17.9%
Fault coverage (%)	72.23%	99.95%	96.2%	50.41%	100%	98.88%
#Test pattern	–	60	54	–	13	22
Max scan length	–	120	12	–	633	42
#scan chain	–	1	13	–	1	16

5. CONCLUSION

We have proposed and implemented an RTL orthogonal scan chain design for the application of sequential circuit testability. A new stack form is designed to reconstruct all the scan chains as possible that can reduce the area and timing overheads effectively than that of other approaches. Extended work will be considered to improve the model of the stack reconstruction methodology of scan chains for upgrading the overhead reduction and fault coverage.

REFERENCES

1. S. M. Reddy and R. Dandapani, "Scan design using standard flip-flops," *IEEE Design and Test of Computers*, 1987, pp. 52-54.
2. C. C. Lin, M. Marek-Sadowska, K. T. Cheng, and M. T. C. Lee, "Test-point insertion: scan paths through functional logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, 1998, pp. 838-851.
3. C. C. Lin, M. Marek-Sadowska, M. T. C. Lee, and K. C. Chen, "Cost-free scan: a low-overhead scan path design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 17, 1998, pp. 852-861.
4. T. Asaka, S. Bhattacharya, S. Dey, and M. Yoshida, "H-SCAN+: a practical low-overhead RTL design-for-testability technique for industrial designs," in *Proceed-*

ings of the IEEE International Test Conference, 1997, pp. 265-274.

5. S. Bhattacharya and S. Dey, "H-SCAN: a high level alternative to full-scan testing with reduced area and test application overheads," in *Proceedings of the IEEE International Test Conference*, 1996, pp. 74-80.

6. C. Aktouf, H. Fleury, and C. Robach, "Inserting scan at the behavioral level," *IEEE Design & Test of Computers*, Vol. 17, 2000, pp. 34-42.

7. R. B. Norwood and E. J. McCluskey, "Orthogonal scan: low overhead scan for data paths," in *Proceedings of the IEEE International Test Conference*, 1996, pp. 659-668.

8. R. B. Norwood and E. J. McCluskey, "High-level synthesis for orthogonal scan," in *Proceedings of the 15th IEEE VLSI Test Symposium*, 1997, pp. 370-375.

9. Y. Huang, C. C. Tsai, N. Mukherjee, O. Samman, D. Devries, W. T. Cheng, and S. M. Reddy, "On RTL scan design," in *Proceedings of the IEEE International Test Conference*, 2001, pp. 728-737.

10. W. S. Chiang, *Efficient VLSI Architectures with Low Hardware Cost for Discrete Wavelet Transform*, Master Thesis, Graduate Institute of Computer and Communication Engineering, National Taipei University of Technology, Taiwan, 2002.

11. R. Andraka, "A survey of CORDIC algorithms for FPGAs," in *Proceedings of the 6th International Symposium on Field Programmable Gate Arrays*, 1998, pp. 191-200.

Chia-Chun Tsai (蔡加春) received the B.S. degree in Industrial Education from National Taiwan Normal University, Taipei, Taiwan, R.O.C., in 1978, and the M.S. and Ph.D. degrees in Electrical Engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1987 and 1991, respectively. From 1978 to 1989, he was a specialist teacher of electronic maintenance at Taiwan Provincial Hsin-Hua and Taipei Municipal Nan- Kang Technology High Schools, respectively. From 1989 to 2005, he served at the Department of Electronic Engineering, National Taipei University of Technology, Taipei, Taiwan, R.O.C.. Since 2005 he has been with the Department of Computer Science and Information Engineering, Nanhua University, Chiayi, Taiwan, R.O.C., where he is currently a Full Professor. From 1994 to 1995 and 2000 to 2001, he was working postdoctoral research at University of California, San Diego, and North Carolina State University, respectively. His current research interests include VLSI design automation and mixed-signal IC design.

Hann-Cheng Huang (黃漢城) received the M.S. degree in Graduate Institute of Computer Communication, and Control from National Taipei University of Technology, Taipei, Taiwan, R.O.C., in 2003. Currently, he is an IC design engineering at the Wellsyn Technology Corporation. His research interests include image processing and IC testing.

Trong-Yen Lee (李宗演) received the Ph.D. degree in Electrical Engineering from the National Taiwan University, Taipei, Taiwan, R.O.C., in 2001. Since 2002, he has been a member of the faculty in the Department of Electronic Engineering, National Taipei University of Technology, where he is currently an Associate Professor. His re-

search interests include hardware-software codesign of embedded systems and FPGA systems design and VLSI design.

Wen-Ta Lee (李文達) was born in Taipei, Taiwan, R.O.C., in 1962. He received the B.S. and M.S. degree in Electric Engineer from National Cheng Kung University, in 1985 and 1989, respectively. He also received the Ph.D. degree in Electric Engineer from National Taiwan University, in 1995. He is an Associate Professor in the Department of Electronic Engineering, National Taipei University of Technology. His research interests are in the areas of VLSI architectures for digital communications, coding theory and signal processing.

Jan-Ou Wu (吳占鰲) received the M.S. degree in Graduate Institute of Computer Communication, and Control from National Taipei University of Technology, Taipei, Taiwan, R.O.C., in 2001. Since 2002, he is working toward Ph.D. at the Graduate Institute of Mechanical Electrical Engineering, National Taipei University of Technology. His current research interests include VLSI design automation and grey theory application.