Abstract
This paper presents a testability improvement method for digital systems described in VHDL behavioral specification. The method is based on testability analysis at register-transfer (RT) level which reflects test pattern generation costs, fault coverage and test application time. The testability is measured by controllability and observability, and determined by the structure of a design, the depth from I/O ports and the functional units used. In our approach, hard-to-test parts are detected by a testability analysis algorithm and transformed by some known DFT techniques. Our experimental results show that testability improvement transformations guided by the RT level testability analysis have a strong correlation to ATPG results at gate level.

1 Introduction
Current VLSI design complexity has made test difficult and expensive. Increasing testability of a design becomes one of the important issues during a design cycle. It helps to achieve high test quality and reduce both test development cost and test application cost.

Many industrial design tools provide VHDL as an input language for hardware description and simulation. Some of them can also perform design synthesis and ATPG to ease design processes. DFT techniques, such as scan path, are sometimes used in these systems for ATPG to achieve high test quality. Using testability measures such as SCOAP [1] to guide the decision of using DFT techniques is proved to be useful [2] and is adopted by Racal-Redac design tool in test generation. Our experience shows that using SCOAP for testability analysis to select test insertion points can efficiently improve the testability. However, this decision made at gate level after the main synthesis tasks are finished is too late for design optimization. The performance and area may not be optimized. In other words, if we can make a decision earlier, we will have more freedom to adjust the decision and optimize the design together with other synthesis tasks in terms of performance and cost. Another disadvantage of this approach is the high computational complexity for gate level testability calculation. This is important since testability will be repetitively calculated during design transformations.

Some structural approaches have been proposed for solving the complexity problem of testability analysis [3,4,5]. In these approaches, flip-flops responsible for feedback loops are selected for scan, so that the remaining circuit is acyclic. It has been shown [3] that feedback loops are the main burden for sequential test generation. However, optimal solutions in these approaches are not guaranteed. Moreover, the analysis of feedback loops requires thorough graph analysis, the complexity of which should also be considered.

Our approach is to raise the level of abstraction for testability analysis. The advantage of our testability analysis approach is the ability to capture hard-to-test parts not only in feedback loops but also in acyclic design parts. We make testability improvement decisions at RT level by using the testability analysis results and some DFT techniques. The early decision about testability improvement gives the possibility that this design can be optimized in later synthesis processes. The testability analysis carried out at high level of abstraction (RT level) will also reduce the computational complexity, since the complexity of a design at that level is significantly lower.

In our approach, a design described in VHDL behavioral specification is compiled into an internal representation used in our high-level synthesis system, called ETPN (Extended Timed Petri Net) [6, 7]. The testability analysis and improvement are carried out on this representation. Our testability analysis algorithm measures the relative difficulties of test generation (cpu time), test application (cycles) and achieving high fault coverage for every line (multi-bit) in a design, which links two RT level functional units, such as adder, multiplier, register and multiplexer. Two strategies are developed for using two DFT techniques to improve testability based on these results. The two DFT techniques...
are partial scan technique and test-cell (T-cell) insertion technique. If there exist registers responsible for the hard-to-test parts, the first strategy will transform these registers into scan registers. Otherwise, the second strategy will add T-cells to hard-to-test lines. The design with these transformations will later be considered together with other high-level synthesis tasks, so that the whole design is optimized. The transformations of scan registers and T-cell insertions can also be annotated back to the original VHDL specification.

The rest of this paper will first briefly introduce the ETPN representation and the VHDL to ETPN compiler. A definition of testability and its algorithm are then given. The testability improvement strategies based on testability analysis results are later described in detail. Finally, experimental results are presented.

2 Design Representation

Our testability analysis algorithm analyzes the ETPN representation of a design and predict hard-to-test parts. We use the S’VHDL compiler [8] to compile a subset of VHDL behavioral specifications into ETPN internal design representations [7]. An example of a VHDL specification and its ETPN representation is given in figure 1.

The ETPN design representation consists of two parts: data path and control part. The data path is a directed graph with nodes and lines. The node represents the storage and manipulation of data. And the line connecting two nodes represents the flow of data. The control part is modelled as a timed Petri net. These two parts are related through the control states in the control part controlling the data transfers in the data path, and the condition signals in the data path (for example, C1 in figure 1) controlling some transition(s) in the control part.

3 Testability Definition

Our testability definition assumes that the fault model is stuck-at fault model and ATPG is random and/or deterministic. The assumptions are justified, since the stuck-at fault model is mostly used fault model and many ATPG’s start by using random test generation to cover as many faults as possible and then switch to deterministic test generation.

The testability is defined by the measures of controllability and observability. The controllability measures the cost of setting up any specific value on a line. The observability, on the other hand, measures the cost of observing any specific value on a line. The term ‘cost’ reflects: 1) the costs of cpu time to find out input vectors to set up a value on an internal line or to distinguishably observe a value on an internal line through primary outputs, 2) the cost of achieving high fault coverage, and 3) the time to control or observe this fault during test execution. The controllability and observability measures reflect respectively two procedures during test generation and test application, namely the fault sensitization and the fault propagation. Both the controllability and the observability are further defined by two factors: combinational factor and sequential factor. The combinational factor is used to measure the cost of generating a test and fault coverage. The sequential factor is used to measure the sequential complexity of repeatedly using a combinational test generation algorithm to a sequential circuit, and the cost (time and memory) of executing a test. As a result, we have four measures for testability: combinational controllability (CC), sequential controllability (SC), combinational observability (CO), and sequential observability (SO).

The combinational factors CC and CO range from 0 to 1, where value 1 represents the best combinational controllability (or observability). The CC’s (or CO’s) of all primary inputs (or outputs) are equal to 1. The value 0 of CC (or CO) represents that we are unable to set up (or observe) that line. The sequential factors SC and SO are natural numbers which represent 1) the estimated number of steps that a combinational test generation algorithm needs to be repeatedly applied to a sequential circuit, and 2) the number of

entity Sum is
port (p1: in bit_vector (0 to 7); p2: out bit_vector (0 to 7));
end Sum;

architecture beh of Sum is
signal x: bit_vector (0 to 7) := “00000000”;
signal y: bit_vector (0 to 7) := “00000000”;
begin
process
begin
wait until y’event;
y <= p1;
while y >= “00000000” loop
x <= x + y;
y <= p1;
wait until y’event;
end loop;
p2 <= x;
end process;
end beh;

Figure 1 (a) VHDL Example, (b) ETPN Example
clock cycles to control or observe a line under test. The SC’s (or SO’s) of all primary inputs (or outputs) are equal to 0.

A set of heuristics is developed to find the controllability (or observability) at a functional unit’s output (or input) based on the controllability (or observability) at its inputs (or outputs). The relationship between the controllability (or observability) at a unit’s input and output depends on the controllability transfer factor $\text{CTF}$ (or observability transfer factor $\text{OTF}$) of a unit. The factor $\text{CTF}$ reflects the probability of setting a value at a unit’s output by randomly exercising its inputs. $\text{OTF}$ reflects the probability of observing a unit’s input by randomly exciting its other inputs and observing its outputs. These two factors reflect not only the difficulties in a random test generation but also the difficulties in a deterministic test generation. The reason is that in deterministic test generations, we need to set a maximum cpu time for each fault and a maximum backup for each vector. Faults with small probability to detect will more often cause conflicts between the vector to control them and the vector to observe them. Therefore, backups are necessary, which means that the probability of detecting the faults becomes smaller within limited cpu time.

Both $\text{CTF}$ and $\text{OTF}$ are in the range from 0 to 1, where 1 represents the best controllability and observability transfer of a unit. The detailed calculation can be found in [9]. Different types of units have different $\text{CTF}$s and $\text{OTF}$s. $\text{CTF}$ and $\text{OTF}$ for each type of units are calculated only once and stored in a library. In the following, we will describe the heuristics of calculating controllability and observability.

### 3.1 Controllability of Combinational Units

Assume that a combinational unit has inputs $X_1, ..., X_p$ with $n_1, ..., n_p$ bits respectively, and outputs $Y_1, ..., Y_q$ with $m_1, ..., m_q$ bits respectively. For combinational units, data transfer from inputs to outputs is finished in the same control state. Combinational ATPG can be applied directly. The controllability at an output of this unit is obtained by

$$C_{C Y_i} = C_{C in} \times \text{CTF}_{U}$$

$$S_{C Y_i} = S_{C in} + \text{clk} (S_i), \ (1 \leq k \leq q)$$

where

$$C_{C in} = \sum_{i=1}^{p} (C_{C X_i} \times n_i)$$

and

$$S_{C in} = \text{Max} \{S_{C X_i}\}$$

$\text{Clk}(S_i)$ is the number of clock cycles needed for data operation at a functional unit (we usually assume it to be zero). The heuristic says that the average $CC$ from a unit’s input to output is reduced because of the data transfer through the unit, and $CS$ is increased because of the delay for data operation.

### 3.2 Controllability of Sequential Units

For sequential units, such as registers, the data transfers from input $X$ to output $Y$ is determined by the related state transitions (from $S_i$ to $S_j$) in the control part. We estimate this by considering the controllability at condition nodes responsible for the state transitions. The $\text{CTF}$ of a sequential unit is one. Thus, we have

$$C_{C Y} = C_{C X} \times C_{C cond} (S_i, S_j)$$

$$S_{C Y} = S_{C X} + \text{clk} (S_i), \ (1 \leq k \leq q)$$

where $C_{C cond}(S_i, S_j)$ is the product of $CC$’s of all condition nodes for state transitions from $S_i$ to $S_j$. It reflects the cost of an ATPG to find an input to make these conditions true.

### 3.3 Observability of Combinational Units

To propagate a fault, we need both to control some input lines and to observe output lines of a unit. The observability at an input line $X_k$ of a unit is obtained by

$$C_{O X_k} = C_{O out} \times \text{OTF}_{U} \times C_{C in}$$

$$S_{O X_k} = S_{O out} + \text{clk} (S_i), \ (1 \leq k \leq q)$$

where

$$C_{O out} = \sum_{i=1}^{q} (C_{O X_i} \times m_i)$$

$$S_{O out} = \sum_{i=1}^{q} (S_{O Y_i} \times m_i)$$

$C_{C in}$ is the average controllability (without $C_{C X_k}$) as defined in section 3.1.

### 3.4 Observability of Sequential Units

The $\text{OTF}$ of a sequential unit is one. Similar to the formulas in 3.2, we have

$$C_{O X} = C_{O Y} \times C_{C cond} (S_i, S_j)$$

$$S_{O X} = S_{O Y} + \text{clk} (S_i, S_j)$$

### 4 Testability Analysis Algorithm

The testability analysis algorithm calculates the testability $\{C, SC, CO, SO\}$ for each line in a design. It first assigns ones to $CC$s and zeros to $SC$s for all primary inputs in the data path of the ETPN. These values are then propagated according to the heuristics described above until primary outputs are reached. A similar approach is used in calculating observability, but it starts from primary outputs towards inputs. A simplified version of the algorithm for $CC$ and $SC$ calculation is presented in the following:

**A1** assign all primary inputs: $CC_{prim in} = 1$, $SC_{prim in} = 0$.

**A2** calculate controllability at a unit’s outputs by using the average controllability calculated at its inputs:
5 Testability Improvement Strategies

Two DFT techniques, the partial scan technique and the T-cell insertion technique are used to improve testability. The former technique can make some registers totally accessible. The latter one can make some lines totally accessible, but is only used when no register is responsible for hard-to-test parts. Two strategies of identifying hard-to-test registers and hard-to-test lines for the uses of these two techniques are developed.

5.1 Sequential Unit Selection Strategy

Experiments [3] show that feedback loops are mainly responsible for test generation complexity. To solve this problem, we break feedback loops by making nodes directly accessible. Partial scan technique which makes some registers directly accessible is relatively cheap and easy. The proposed sequential unit selection strategy uses this technique to transform registers with the worst testability and involved in feedback loops to scan registers. It first evaluates the testability of registers by the formula:

\[ E_{R_i} = \frac{C_{R_i} - C}{C} + k \cdot \frac{S - S_{R_i}}{S} \]

where \( C_{R_i} \) and \( S_{R_i} \) are \( C_{R_i} + S_{R_i} \) at the output of register \( R_i \) respectively. \( C \) and \( S \) are the average of all \( C_{R_i} \) and the average of all \( S_{R_i} \) in a design. \( k \) equals \( C/S \), which scales the combinational and sequential factors to the same level. This formula measures the sum of combinational and sequential testability derivations at register \( R_i \) from the average testabilities of all registers. The larger \( E_{R_i} \), the better the testability of register \( R_i \).

After the evaluations, this strategy suggests several registers (about 10% from the beginning) with the worst testability evaluation results and involved in feedback loops to be transformed to scan registers first. It also requires that these registers must be in different feedback loops, since once a register in a feedback loop becomes directly accessible, the testability of the whole feedback loop will be improved. After the first step, testability analysis and evaluation are re-calculated. If the testability evaluations of some registers are still very poor, it will transform them in the next step no matter if they are involved in the same feedback loop as the registers transformed before. If no register is involved in feedback loops, this strategy only transforms one register with the worst evaluation result every time, since testability dependency may exist between registers.

This strategy is usually used at the beginning. When no register is responsible for the bad testability of a design, the following line selection strategy can be used.

5.2 Line Selection Strategy

The line selection strategy uses a similar formula to evaluate the relative difficulty of testing each line in a design. We only need to change \( C_{R_i} \) and \( S_{R_i} \) for registers in the above formula to those for lines. This strategy suggests the insertion of a T-cell in the line with the worst testability. The T-cell is a scan-like element designed to enhance controllability and observability at any line in a design. The detailed schematic design can be found in Racal-Redac system user’s manual. This strategy only selects one line for transformation each time because of testability dependencies between relevant lines. Testabilities at these relevant lines must be re-calculated after each transformation.

6 Experimental Results

To verify the correlation between our testability analysis results at RT level and test generation results at gate level, we use the Racal-Redac design tool to synthesize RT level designs with different testability improvement transformations to gate level (by SilcSyn) and then do test generation (by Intelligen). Comparing the test generation results, we
can find the influence of our testability improvement transformations guided by the testability analysis results at RT level to the test generation results at gate level. The input designs are specified in behavioral VHDL and the partial scan transformations are specified in VHDL attributes that

```vhdl
entity Cnt is
  port ( 
    clk : in std_logic;
    reset : in std_logic;
    x : in std_logic_vector (7 downto 0);
    max : out std_logic_vector (7 downto 0);
    sum : out std_logic_vector (15 downto 0);
  );
  attribute clock_source of clk : signal is true;
  attribute reset_source of reset : signal is true;
end Cnt;
```

```vhdl
architecture beh of Cnt is
  signal c : std_logic_vector (7 downto 0);
  signal m : std_logic_vector (7 downto 0);
  signal s : std_logic_vector (15 downto 0);
  attribute rts of c : signal is include; -- if c needs to be scanned
  attribute rts of s : signal is include; -- if s needs to be scanned
begin
  max <= m;
  sum <= s;
  count: process
  begin
    wait until (reset = '1');
    resetloop: loop
      if reset = '1' then -- initialization.
        c <= "11111111";
        m <= "00000000";
        d <= "0000000000000000";
      end if;
      wait until rising_edge(clk) or (reset = '1');
      next resetloop when (reset = '1');
      while c /= 0 loop
        c <= c - 1;
        if (x > m) then
          m <= x;
        end if;
      s <= s + x;
      wait until rising_edge(clk) or (reset = '1');
      next resetloop when (reset = '1');
      end loop; -- while loop.
      end loop; -- resetloop.
  end process;
end beh;
```

Figure 3 Counter SilcSyn VHDL Specification

<table>
<thead>
<tr>
<th>register in design</th>
<th>testability evaluation (E_{RT})</th>
<th>fault cov. (%)</th>
<th>test appl. (cycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td>no scan</td>
<td>no</td>
<td>595</td>
<td>94.72</td>
</tr>
<tr>
<td>C (8 bits)</td>
<td>selected</td>
<td>868</td>
<td>92.83</td>
</tr>
<tr>
<td>M (8 bits)</td>
<td>no</td>
<td>2,821</td>
<td>95.31</td>
</tr>
<tr>
<td>S (16 bits)</td>
<td>no</td>
<td>5,030</td>
<td>90.95</td>
</tr>
<tr>
<td>CC+CO</td>
<td>no</td>
<td>169</td>
<td>90.42</td>
</tr>
<tr>
<td>SC+SO</td>
<td>no</td>
<td>2,819</td>
<td>96.30</td>
</tr>
<tr>
<td>C, S, M</td>
<td>full scan</td>
<td>931</td>
<td>96.48</td>
</tr>
</tbody>
</table>

SilcSyn can accept. The experimental framework is illustrated in figure 2.

The experiment was carried out on a Sparc-10 station and the default setting of maximum cpu time for each fault (500 seconds) and maximum backup for each vector (25) were used. Designs used for experiments are three high-level synthesis benchmarks and one illustrative example, counter.

Figure 3 describes the SilcSyn VHDL specification for the counter example. Table 1 lists its testability evaluation results when the sequential selection strategy is used. The testability evaluation results show that register C has the worst testability evaluation value 0.20. In fact, register C which controls the iteration of the counter is uncontrollable. After register C is inserted into a scan path, the average CC+CO \(0 \leq CC+CO \leq 2\) is improved from 0.4 to 1.59. The average SC+SO \((\geq 0)\) is improved from 7,707 to 7. By a similar step, register S is selected for scan insertion, and CC+CO is improved from 1.59 to 1.68, not as much improvement as in the first step. On the other hand, SC+SO becomes worse, from 7 to 13. This is explained by the fact that with more registers scanned, the delay in test application caused by the scan path starts to increase.

Table 2 lists Intelligent test generation results of all different scan transformation schemes for this design. Our partial scan scheme with registers C and S scanned achieves much shorter cpu time of test generation (76 seconds) and the best fault coverage (97.08%). In full scan case, the added logic for scanning register M does not help test, since it is easy to access. On the contrary, the unnecessary added logic increases the test generation complexity. As a result, the fault coverage is decreased and the test application cycle is increased.
Table 3 shows the SilcSyn synthesis result and Intelligen test generation results with different test transformation schemes: no scan, full scan and our partial scan for the Differential-equation benchmark [11], the Square-root benchmark [12], and the Elliptic-filter benchmark [13]. We can see the test improvement by our partial scan transformation in test generation time and fault coverage when comparing with no scan, and the improvement in hardware overhead (number of gate) and test application time when comparing with full scan design. These results conform well to our testability predictions at RT-level.

### Table 3: ATPG Results with Different Scan Schemes

<table>
<thead>
<tr>
<th>bench-mark</th>
<th>scan scheme</th>
<th>synthesis</th>
<th>ATPG</th>
<th>no. of</th>
<th>cpu time</th>
<th>fault cov.</th>
<th>test appl.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bits scanned</td>
<td>no. of gates</td>
<td>cpu time (s.)</td>
<td>fault cov. (%)</td>
<td>test appl. (cycle)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diff. Eq.</td>
<td>no (0)</td>
<td>3.269</td>
<td>12.915</td>
<td>49.68</td>
<td>6.224</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial (32)</td>
<td>3.398</td>
<td>3.937</td>
<td>94.17</td>
<td>18.345</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>full (144)</td>
<td>3.687</td>
<td>4.97</td>
<td>98.05</td>
<td>108.308</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Square Root</td>
<td>no (0)</td>
<td>2.746</td>
<td>8.014</td>
<td>32.76</td>
<td>3.266</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial (64)</td>
<td>2.899</td>
<td>11.77</td>
<td>92.85</td>
<td>53.773</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>full (80)</td>
<td>2.942</td>
<td>1.011</td>
<td>91.72</td>
<td>51.344</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Elliptic Filter</td>
<td>no (0)</td>
<td>8.835</td>
<td>72.852</td>
<td>6.55</td>
<td>2.901</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial (64)</td>
<td>8.646</td>
<td>61.782</td>
<td>54.77</td>
<td>79.280</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial (144)</td>
<td>8.974</td>
<td>35.343</td>
<td>79.88</td>
<td>280.271</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>partial (272)</td>
<td>9.495</td>
<td>31.993</td>
<td>88.35</td>
<td>887.234</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>full (576)</td>
<td>10.736</td>
<td>11.442</td>
<td>96.10</td>
<td>964.589</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 7 Conclusion and Discussion

A testability improvement method based on testability analysis for designs described in VHDL behavioral specification is presented. This approach is able to detect and improve hard-to-test parts not only in feedback loops but also in acyclic parts, which other structural approaches are not applicable. The computational complexity is also much lower than that at gate level. The early testability improvement decision makes the optimization of the testability-improved design possible in other synthesis processes, and the test development (test generation, etc.) much easier.

The testability improvement techniques discussed in the paper are partial scan technique and T-cell insertion technique. This means that we are able to improve testability not only for registers, but also for any other parts in a design.

Our experiments with high level synthesis benchmarks show that designs without test consideration will cause huge cpu time for test generation and very low fault coverage. The full scan technique does not always solve these problems. Not only the long delay in test application and high hardware overhead, the added logic for full scan may also increase the test generation complexity.

We are trying to detect gate level test problems by testability analysis at RT level. This is under the assumption that we estimate CTFs and OTFs of RT level functional units only by their functional tables, since we do not know their detailed implementations at that level. We can calculate them more accurately if we know their implementations.

In this paper, we have discussed the usage of two DFT techniques to improve testability. The use of other DFT techniques, such as BIST, is our on going research.

### Acknowledgement

Many thanks to Staffan Persson from CynCrona Redac in Sweden who helped us in using the Racal-Redac system for experiments.

### References