

SYNTHESIS OF TESTABLE LOGIC FROM RTL VHDL

Alison A. Li
LSI Design Engineer
Motorola Inc,
Government and Systems Technology Group
Scottsdale, Arizona 85252

ABSTRACT

The goal in a design environment is to reduce cycle time by efficient reuse of previously generated circuitry. VHDL provides an efficient means to achieve these goals. RTL VHDL can be used to generate efficient logic designs in the ASIC environment. In some situations, reuse of design work may involve merging synthesized logic designs with circuitry previously designed "by hand." Care must be taken when generating the RTL VHDL to fully describe the interface between the synthesized gates and the existing logic design. Failure to fully describe this interface may result in unnecessary logic that may impact on the degree of testability of an ASIC design.

INTRODUCTION

VHDL is a tool, when used properly, that can significantly decrease the design cycle of large scale integrated circuits. The circuitry of a well-designed ASIC must not only meet all functional requirements, but must also be testable. Most ASIC vendors require test patterns to be provided with the design. If functional test patterns have been generated, it is usually sufficient that these patterns

detect a minimum of 95% of single stuck-at faults. The degree of fault coverage can be affected by the presence of redundant gates. Test compilers can generate test patterns that can detect a large percentage of stuck-at faults at the expense of adding scan chains throughout the design. However, with large complex designs, ASIC designers cannot always afford the added gate area necessary to implement test scan chains.

The typical fault grade process involves applying a set of test patterns, checking the results, then modifying the functional test patterns to cover the undetected nodes. The problem with synthesis-generated schematics is that the process of making the correlation between undetected nodes and the RTL VHDL is difficult. A correlation between the RTL VHDL and the synthesized schematics must be made to determine how best to improve the functional fault grade patterns. In some cases, nodes may be undetectable since synthesis of the RTL VHDL generated gates may add unnecessary functionality to certain modules.

If VHDL is used to synthesize gates, how can the designer know if unnecessary gates, which can adversely impact the fault grade coverage, are present in the design? If unnecessary gates are present, why were they

synthesized, and how can the VHDL code be modified to reduce the possibility of creating these gates? The answers to these questions can only be found after careful examination of the synthesized gates and VHDL code. Some examples of how unwanted gates can easily be inferred in the design are described in this paper. A comparison between the original RTL VHDL and the RTL VHDL descriptions after slight modifications have been made are also presented.

VHDL SYNTHESIS EXPERIENCE

A significant decrease in the design cycle of an ASIC device has been achieved with the use of both “traditional” design techniques mixed with the RTL design methodology. This approach was optimized to reduce the overall design cycle time by combining the benefits of each technique.

RTL VHDL descriptions were used for much of the design of a highly complex math co-processor ASIC. The design of the co-processor involved the integration of existing circuitry that implemented a math algorithm with new control circuitry. The math processor core had been previously designed, and had been optimized by hand for maximum speed and minimal gate count. The task was to create control circuitry that could configure this math core to perform many different math functions. An additional requirement was to allow several different microprocessors to asynchronously access the ASIC.

Since the ability for VHDL synthesis is well proven for synchronous design, it was decided to generate all synchronous control logic through synthesis. The asynchronous interface was designed using the “traditional” method of entering logic gates by hand with a schematic capture tool. The overall design was approached top down, by breaking down the design into small functional blocks, and

entering these block designs using a schematic capture tool. The functional descriptions of these hierarchical blocks were written in RTL VHDL and synthesized. Using synthesis to generate all gate level designs eliminated the need for time-consuming and tedious gate level schematic capture.

After the design was proven to be fully functional, a preliminary set of functional test patterns was used to run fault grade. The fault grade process checks every node in the circuit, one at a time, to determine whether a stuck-at-one and stuck-at-zero fault can be detected at the output pins of the ASIC. The initial patterns were written to target specific functional blocks within the design. The results of the fault grade were very poor. Upon examination of the undetected nodes, and making a correlation between the synthesized gates and the RTL VHDL, it became apparent why many of the nodes were undetectable. Synthesis of the RTL VHDL created logic that was fully functional, but also provided circuitry for specific conditions that would never occur in the system. Test patterns could never cause these conditions to occur, and therefore the circuit could not be fully tested. Upon closer examination of the RTL code, it was evident that the synthesis tool was correct to generate these gates. The RTL code was revisited to see if it could be written to synthesize without these unnecessary gates.

The majority of the undetected nodes were within synthesized blocks of random logic, at the interface between the synthesized gates and the existing logic. Structured designs such as counters or registers were synthesized effectively and achieved a satisfactory fault grade score. Most of the RTL VHDL written to describe the combinational logic was written in the form of IF or CASE statements. Care was taken to avoid latches by fully specifying the output. A default condition was stated to cover any


```

-- default conditions
OUT1 <= '1'; OUT2 <= '1'; OUT3 <= '1';
If (ENABLE = '1') then
  If (IN1 = '1') then
    OUT1 <= '1';
  end if;
  If (IN2 = '1') then
    OUT2 <= '1';
  end if;
  If (IN3 = '1') then
    OUT3 <= '1';
  end if;
end if;

```

For the case where no two outputs will be active high at the same time, this code is functionally the same as the original code. However, the revised RTL description implies that all outputs have equal priority. Each output is now generated by ANDing the ENABLE signal with the appropriate input control signal. The schematic synthesized with the modified RTL VHDL is shown in Figure 1b. Note that the resulting circuit is more efficient, and that the input decoding circuitry was not synthesized. The same fault grade patterns were able to detect 100% of the stuck-at faults in the resulting synthesized schematic.

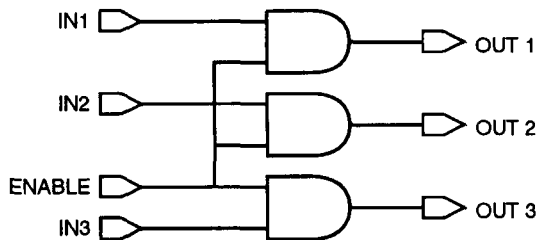


Figure 1b. Example 1, Smaller IF Statements

Example 2

In a bus-oriented design, it may be necessary to create a large multiplexer that routes a particular input bus to the output based on input control signals. In this example, an input control bus determines the bus selection. Each bit position of the control bus

selects a particular input bus to route to the output. It can be assumed that no two busses will be selected at the same time. This circuit may be described in RTL VHDL using a CASE statement as follows:

```

Case CTL_BUS is
  when "001" =>
    OUT_BUS <= A;
  when "010" =>
    OUT_BUS <= B;
  when "100" =>
    OUT_BUS <= C;
  when others =>
    OUT_BUS <= "00";
end case;

```

This logic selects input bus A when the bit 0 of the input control bus, CTL_BUS, is active high. Bus B is selected when bit 1 of CTL_BUS active high. Bus C is routed to the output when bit 3 of the CTL_BUS is active high. As seen in Figure 2a, the circuit synthesized from this RTL description includes logic that decodes the input conditions specified. However, since no two bits in the control bus would be high at the same time, this decoding is unnecessary, and this logic cannot be fully tested.

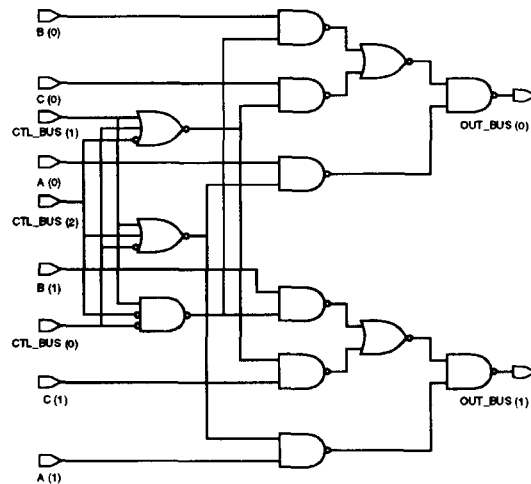


Figure 2a. Example 2, Case Statement

A fault grade pattern that selects one bus at a time was applied to the synthesized logic. Much of the decoding logic was undetectable by the fault grade patterns.

To remove the input decoding circuitry, the code may be rewritten as separate IF statements. The outputs are now defined in the RTL code as a function of only one input control bit. The revised code is as follows:

```
-- default
OUT_BUS <= "00";
If (CTL_BUS(0) = '1') then
    OUT_BUS <= A;
end if;
If (CTL_BUS(1) = '1') then
    OUT_BUS <= B;
end if;
If (CTL_BUS(2) = '1') then
    OUT_BUS <= C;
end if;
```

These three IF statements describe different conditions of the same output. The circuit synthesized from this RTL code is shown in Figure 2b. Note that this RTL VHDL description synthesized fewer gates to perform the same function. However, as in Example 1, the RTL VHDL is interpreted by the synthesis compiler as having levels of priority. If CTL_BUS(0) is active, then the output bus is set to A. Since further statements that affect the output follow the first IF statement, the output bus is set to A only if the remaining bits of the input control bus are not active. Similarly, the output bus is equal to B only if CTL_BUS(1) is active high and CTL_BUS(2) is inactive low. Because of the order of the IF statements, the value of bit(0) of CTL_BUS does not matter. The output bus is set to bus C if CTL_BUS(2) is active high, regardless of the value of the other control bits. The last statement to affect the output value has the highest priority in the decode logic. If no two input control bits can be active at the same time, there is again unnecessary decoding of the input signals. Again, the fault grade

patterns cannot test the decode logic, leaving undetected faults.

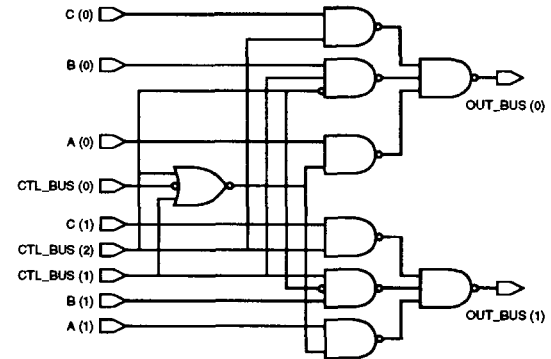


Figure 2b. Example 2, IF Statements

Optimal synthesis was achieved with the following RTL description:

```
-- default conditions
A_BUS <= "00"; B_BUS <= "00";
C_BUS <= "00";
If (CTL_BUS(0) = '1') then
    A_BUS <= A;
end if;
If (CTL_BUS(1) = '1') then
    B_BUS <= B;
end if;
If (CTL_BUS(2) = '1') then
    C_BUS <= C;
end if;
OUT_BUS <= A_BUS or B_BUS or
    C_BUS;
```

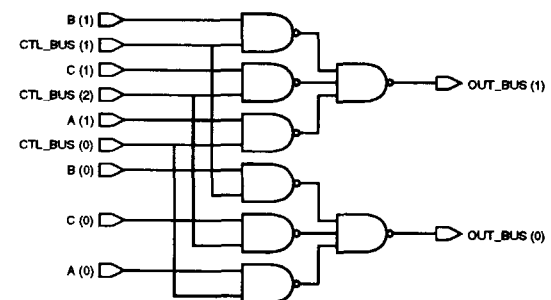


Figure 2c. Example 2, Implied Structure

This code is highly structured, and roughly infers the structure of the synthesized AND and OR gates. No unnecessary decoding of the input control signals is performed, and the synthesized logic, shown in Figure 2c, reflects the desired function. The schematic synthesized from this RTL description obtained a fault grade score of 100%.

Example 3

An output decoder may be described in many different ways. Given discrete inputs to decode into an output bus, it may seem reasonable to write the logic description as follows:

```
-- default condition
OUT_BUS <= "000";
If (IN1 = '1') then
  OUT_BUS <= "011";
end if;
If (IN2 = '1') then
  OUT_BUS <= "101";
end if;
If (IN3 = '1') then
  OUT_BUS <= "110";
end if;
```

The schematic synthesized from this RTL VHDL is shown in Figure 3a. The synthesized logic that results is a combination of several factors. As in the previous examples, the input control signals have been decoded to create priority where input IN3 has the highest priority and input IN1 has the lowest priority. The output conditions are also “over specified.” A default condition was declared to take care of the case where none of the input signals are active. The full output bus was defined for each input condition. Extra logic was synthesized to ensure that the output bits are low when specified. For example, logic is added to force bit 0 of OUT_BUS low when IN3 is high. This circuitry is not needed since the default condition is sufficient to define this output bit in this case. It is highly

unlikely that fault grade patterns will detect stuck-at faults on logic that forces the output to the same logic level as the default condition.

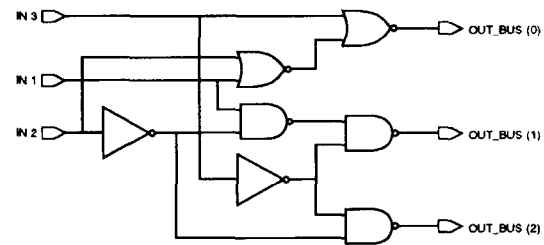


Figure 3a. Example 3, IF Statements

Optimal synthesis was obtained by specifying only the output bits that are different from the default condition.

```
-- default condition
OUT_BUS <= "000";
If (IN1 = '1') then
  OUT_BUS(1) <= '1';
  OUT_BUS(0) <= '1';
end if;
If (IN2 = '1') then
  OUT_BUS(2) <= '1';
  OUT_BUS(0) <= '1';
end if;
If (IN3 = '1') then
  OUT_BUS(2) <= '1';
  OUT_BUS(1) <= '1';
end if;
```

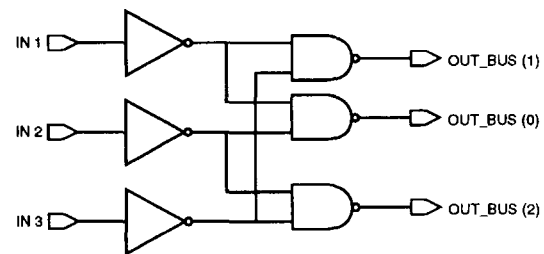


Figure 3b. Example 3, Implied Structure

This RTL code fully describes the output bits. However, with this RTL description, it

is more difficult to determine the value of the output bus since the entire bus was not described. The circuitry synthesized with this RTL description is shown in Figure 3b, and achieved 100% fault grade coverage.

FAULT GRADE EXPERIENCE

For the math co-processor ASIC synthesized on this effort, the initial fault grade score was very poor. By making a few modifications to the RTL VHDL (and by adding a few test vectors), the fault grade score was improved to the desired coverage with no added test logic.

The presence of redundant gates increase the total number of gates in the design, which may lead to an unnecessarily large ASIC. Statistics were taken on one functional block before and after the code rework. Along with the increase in fault grade score, a 6% reduction in gate count was achieved. The benefit of gate reduction alone may justify writing more efficient RTL descriptions.

SUMMARY

These suggestions should minimize the number of nodes undetected by fault grade patterns. The coding examples create unnecessary decoding and unused gates that do not add to the function of the circuit. However, to determine whether gates need not be fault graded, the synthesized gates must be examined thoroughly. This defeats the purpose of designing with VHDL, where gate level debug should be kept to a minimum. Since a test path compiler does not generate functional patterns, the compiler may be able to generate additional fault grade patterns to detect these nodes at the expense of a large number of unnecessary test vectors.

By making a few modifications to the code, a more efficient synthesis can be obtained that results in a better fault grade score. The last two cases show that when more structure is inferred in the RTL descriptions, more optimal synthesis can be obtained. Optimal synthesis can be obtained when the RTL coding is approached with a hardware implementation in mind. The decoding inferred in the different coding styles must be kept in mind. These examples have shown that it is easy to infer unwanted logic that can have a detrimental effect on the fault grade score and the efficiency of the resulting gates.