A Methodology for Automated Insertion of Concurrent Error Detection Hardware in Synthesizable Verilog RTL

Kartik Mohanram, C.V. Krishna, and Nur A. Touba

University of Texas at Austin
Austin, TX 78712-1084

ABSTRACT

This paper describes a methodology for automated insertion of concurrent error detection (CED) circuitry in synthesizable Verilog RTL that allows easy tradeoff between overhead and coverage. The insertion is done at the front-end of the synthesis process which is highly advantageous. Computer-aided design (CAD) tools that support the proposed methodology have been implemented and are described. The CAD tools allow the designer to select from different CED schemes that provide various cost/coverage tradeoffs. The CED circuitry is then automatically inserted into the Verilog design. Experimental results are shown which demonstrate the broad range of design points that the proposed methodology offers the designer to choose from to satisfy cost/coverage requirements. The methodology can be seamlessly integrated into the typical design flow used in industry.

1. INTRODUCTION

The move towards deep-submicron VLSI technologies with higher frequencies, lower voltage levels, and smaller noise margins is increasing the susceptibility of systems to transient and intermittent faults resulting in "soft errors." Early detection of errors is crucial for preserving the state of the system and maintaining data integrity. Circuit-level techniques for concurrent error detection (CED) permit early detection and containment of errors before they can propagate to other parts of the system and corrupt data. This reduces the complexity and increases the effectiveness of system-level fault tolerance features.

The general approach for performing CED with a systematic error detecting code is illustrated in Fig. 1. The function logic is augmented with a check symbol generator. The function logic has n output bits, and the check symbol generator provides k check bits. Together, they form an (n,k) systematic code. There is a checker which monitors the encoded outputs and gives an error indication if a non-codeword occurs. The checker can be designed so that it is *self-checking*, which means that it is guaranteed to give an error indication if a fault occurs in the checker itself.

Efficient schemes have been developed for CED in circuits with regular structures, e.g., adders [Lo 92], [Gorshe 96], multipliers [Pradhan 86], PLAs [Mak 82], [Nicolaidis 89], etc. However, the problem is much more difficult for control logic which typically has an irregular multilevel structure [Touba 97].

As process technology scales below 100 nanometers, current studies indicate that circuits will become increasingly sensitive to noise, particularly to atmospheric neutrons and alpha particles. It is projected that this will result in unacceptable soft error rates even in mainstream commercial electronics.

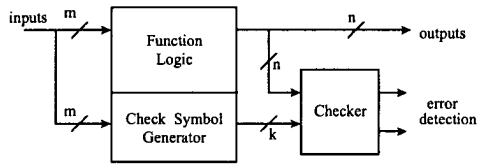

Figure 1. General Approach for Internal Error Detection

Previous research in CED has focused on mission critical systems where very high levels of dependability are required. In these applications, the main goal is to provide high levels of coverage, while the cost of the error detection circuitry is of less importance. As CED becomes increasingly important in more mainstream commercial electronics, there is a need for more cost-effective techniques. The two main sources of cost in incorporating error detection circuitry are:

1. Design time – The time required to properly insert the error detection circuitry in a design and to verify that it works correctly and meets coverage requirements.
2. Overhead – The error detection circuitry requires additional silicon area and can impact the performance, power consumption, and other design criteria.

There is a need for computer-aided design (CAD) tools and methodologies to automate the process of inserting error detection circuitry and to allow easy tradeoffs between overhead and coverage.

In this paper, we describe a methodology for automated insertion of CED in synthesizable Verilog RTL that allows easy tradeoff between overhead and coverage. This methodology was developed as part of a joint project between researchers at the University of Texas-Austin and Hewlett Packard.

A common design flow that is used at Hewlett Packard and elsewhere is for a designer to describe a design in synthesizable Verilog RTL and then use a synthesis tool to generate an implementation. An advantage of having a Verilog RTL description of the design is that it can be used for fast RTL simulation, and it provides a high-level, easy to understand documentation of the design's function. We insert the CED circuitry at the RTL level rather than at the gate level because insertion at the front-end of the synthesis process has several advantages. It allows the error detection circuitry to be optimized along with the functional circuitry. The synthesis tool can take the error detection circuitry into account when satisfying timing constraints (as well as other constraints on power, testability, etc.). This avoids problems with having to reiterate the synthesis process because back-end insertion of error detection circuitry caused timing or other constraints to be violated. The approach of inserting the error detection circuitry at the RTL level can be easily and seamlessly incorporated into the standard design flow.

0-7803-7448-7/02/$17.00 ©2002 IEEE

We have developed CAD tools to support this methodology and have evaluated it on existing Verilog designs at Hewlett Packard. This paper describes the methodology and discusses the results that have been obtained.

2. OVERVIEW OF METHODOLGY

A flowchart for the methodology is show in Fig. 2. Given a synthesizable Verilog description, the Verilog is parsed and the CED circuitry is inserted in the design at the appropriate locations. The CED scheme to be used is selected by the designer. Different schemes can be used depending on the coverage/cost requirements. After the CED circuitry has been inserted in the design, the resulting Verilog description can then be passed to a commercial synthesis tool. The synthesis tool generates a gate-level implementation. Fault simulation is then performed on the gate-level implementation to evaluate the coverage that is provided by the CED circuitry. Other design criteria such as area, delay, power, etc., for the implementation can also be evaluated. If the coverage that is provided by the CED circuitry meets requirements and the other design criteria are also satisfactory, then the process is complete. However, if the coverage or cost is not satisfactory, then the designer can repeat the process with a different CED scheme. Thus, this methodology supports rapid evaluation of the design space to converge on the best implementation.

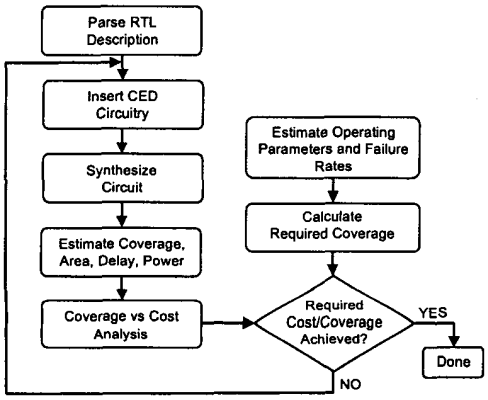

Figure 2. Flowchart for Methodology

We have developed two CAD tools to support this methodology. The first is a CED insertion tool. This tool takes as an input a synthesizable Verilog description, and automatically inserts the CED circuitry based on the options specified by the designer. The tool outputs a modified synthesizable Verilog description with the appropriate CED circuitry in place. The modified Verilog description can then be used for RTL simulation and passed on to the synthesis tool. The second CAD tool is a coverage evaluation tool. This tool takes as an input the gate-level implementation after synthesis and evaluates the coverage that is provided by the CED circuitry.

The techniques used in the CED insertion tool and coverage evaluation tool are described in Sections 3 and 4.

3. CED INSERTION TOOL

The goal of the CED insertion tool is to provide the designer with various options in terms of which CED schemes to use, and then automatically insert the error detection circuitry into an existing synthesizable Verilog design. Synthesis is generally not used for generating an implementation for the data path portion of a design. The data path logic has a regular structure and is generally constructed from highly optimized components. For these components which have a regular structure, very efficient CED schemes exist (e.g., self-checking adder or multiplier designs [Lo 92], [Gorshe 96]). Synthesis is generally used for obtaining an implementation of the control logic portion of a design which has an irregular multilevel structure. The CED insertion tool described here is targeted towards the control logic portion of a design.

3.1 CED Schemes Supported

The control logic consists of finite state machines (FSMs) which control the data path. The CED insertion tool must insert error detection circuitry to detect errors in these FSMs. An FSM consists of next-state logic and output logic. Our CED insertion tool provides the designer with four different options in terms of the CED scheme that is used for each FSM:

1. Duplication – In this scheme, the FSM is simply duplicated and the outputs are compared with an equality checker. The area overhead incurred is greater than 100%. If the FSM has a lot of outputs, the equality checker can become very large.
2. One-Hot – In this scheme, a "one-hot" state assignment is used for the FSM (i.e., each state is encoded such that only one bit is a '1' while the rest are all '0'). A one-hot checker is used to check the present state and a parity checker may be used to check the outputs. One limitation of applying this scheme to an existing design is that the states need to be re-encoded. If the design uses special status bits or decodes a subset of the bits in the state register for some operations, then this scheme cannot be applied.
3. Parity – In this scheme, a parity bit is appended to the state encoding. Unlike the one-hot scheme, the states do not need to be re-encoded. Another advantage is that if parity prediction is used on the outputs, then a single parity checker can be used to check both the present state and output parity. This is done by generating the parity check bit so that it is the XOR of both the present state parity and output parity. The advantage of the parity scheme is that it requires less overhead than the duplication scheme, but this comes at the cost of losing coverage of errors that affect an even number of bits. However, in some designs, this loss of coverage is not substantial.
4. Hybrid Parity – In this scheme, a parity bit is added to the state encoding, however, the output logic is duplicated to provide higher error coverage. This approach is a hybrid between duplication and parity. It has less overhead than using full duplication, but better coverage than parity.

Each of the CED schemes has advantages and disadvantages. The designer can choose the scheme that is best suited for each FSM in the design. For FSMs with a small number of states, the one-hot scheme may provide better coverage at lower cost than parity. For larger FSMs, parity may be the most cost effective. If the coverage that is provided by parity is not sufficient, than hybrid parity or duplication can be used.

3.2 Verilog Modification

The CED insertion tool automatically modifies an existing synthesizable Verilog design to insert the CED circuitry. This is done by having the user identify each FSM and specify the CED scheme that is to be used for that FSM. The FSMs are identified by specifying the module name and the present state and next state variables. The tool then parses the module, modifies the

statements containing the present state and next state variables accordingly, and adds the necessary checkers.

For the duplication scheme, the tool simply duplicates the entire FSM and adds an equality checker. For both the parity and one-hot schemes, the Verilog source has to be modified in places where the present and next state variables occur. This is explained in greater detail with respect to the example presented in Fig. 3:

1. Register declarations - The present and next state variables' register declarations are modified appropriately by increasing the width of the register to reflect a parity or one-hot encoded state. In the example in Fig. 3, the parity scheme requires that the present and next state registers change from 2 to 3 bit-wide registers. The parity bit is always the most significant bit, and the value is chosen to reflect an odd parity under normal operation. In a similar manner, the one-hot scheme would require that the registers be modified to be 4 bit-wide registers. Obviously, modifying the register width for the one-hot scheme can prove to be wasteful if all the states are not used in the FSM. Our CED insertion tool provides the user with an option to specify the exact number of states that are used in the FSM so that the present and next state registers are expanded to the optimum width when the one-hot encoding scheme is inserted.
2. Equality operator - The present state variable can also occur in conditional statements, for example, when the equality operator is used to check that the FSM is in a certain state. Here, the state encoding provided in the original design has to be appropriately modified to reflect the parity or one-hot encoding scheme. In the example, 2'b10 is modified to 3'b010, and the parity bit is made the most significant bit. Similarly, 2'b10 would be expanded to 4'b0100 if the one-hot encoding scheme were chosen.
3. Blocking Assignments - The next state variable can occur in blocking assignments when the FSM changes state depending on the combination of inputs and the present state. The changes to the Verilog source are similar to the ones described above for conditional statements.

```
reg [ 1:0 ] PS;                          reg [ 2:0 ] PS;
reg [ 1:0 ] NS;                          reg [ 2:0 ] NS;
assign sigX = ( sigY & ( PS == 2'b10 ) );  assign sigX = ( sigY & ( PS == 3'b010 ) );
always @ ( PS or reset or ... ) begin    always @ ( PS or reset or ... ) begin
   if ( reset ) NS = 2'b00;                 if ( reset ) NS = 3'b100;
   else begin                               else begin
      case ( PS )                              case ( PS )
        2'b00:                                   3'b100:
          NS = 2'b01;                              NS = 3'b001;
          ...                                      ...
        default:                                 default:
          NS = 2'b11;                              NS = 3'b111;
      endcase                                  endcase
   end                                      end
end                                      end
```

Figure 3. Example of Modifying Verilog Design

4. Output Declarations - The parity encoding scheme has significant advantages over the one-hot encoding scheme in two cases. The first is when the contents of the present state register are exported ouside the module in which the FSM is contained, by declaring all (or part) of the present state register as primary outputs of the module. The second case is when all (or part) of the present state register's contents are accessed in say a blocking assignment statement within or outside the module of occurrence of the FSM. In both these cases, the addition of the parity bit in the most significant position does not alter the functionality in any manner, and automation of the parity scheme poses no difficulties. In other words, since the parity encoding scheme is a separable encoding scheme, the modifications to the Verilog source are transparent to the entire design. Another advantage of parity encoding under these circumstances is that the functionality cannot be inadvertently altered during a post-CED injection optimization phase.

Lastly, when more than a single FSM occurs in the design, the error indication signals are stitched together during the write-back phase of the parser, and finally result in two global error indication signals that are encoded in a two-rail code. While the checker for the parity encoding scheme is straight-forward to introduce, care has to be taken to ensure that the checker for the one-hot encoding scheme is self-testing.

4. COVERAGE EVALUATION TOOL

The goal of the coverage evaluation tool is to evaluate the coverage that the CED circuitry provides. This is done at the gate-level after synthesis. The synthesis tool generates a gate-level implementation of the design which contains the error detection circuitry.

We implemented the coverage evaluation tool using a commercial fault simulator. The coverage evaluation tool first generates scripts for the commercial fault simulator. The commercial fault simulator is then invoked to execute the scripts and produce output files. The coverage evaluation tool then parses the output files and computes statistics about the coverage that the error detection tool provides.

There are two options for specifying the input patterns that are used by the coverage evaluation tool. One is for the user to supply the set of input patterns. The other is for the tool to internally generate a set of random input patterns.

The coverage evaluation tool performs three runs of fault simulation. On each run, all single stuck-at faults are fault simulated. If the fault effect for each single stuck-at fault is propagated to an observation point, then the fault is considered detected. The location of the observation points is different for each of the runs. This is necessary to discount all faults that are within the checker circuitry itself, and hence propagate only to the error indication signals and not to the functional outputs. We use the fault dictionary option of the commercial fault simulator to report all the faults that are propagated to the observation points on each run. On the first run, both the error indication signals and the functional outputs are considered as observation points. On the second run, only the functional outputs are considered as observation points. Finally, on the third run, only the error indication signals are considered as observation points.

Let f1, f2 and f3 be the faults propagated to the observation points on the first, second and third runs respectively. The coverage evaluation tool computes the fault coverage as $(f3 - (f1 - f2)) / f2$. Note that $(f1 - f2)$ is the set of faults that propagate errors only to the error indication signals and not to the functional outputs. These are faults that occur in the error detection circuitry itself. $(f3 - (f1 - f2))$ gives the set of faults that cause errors in the functional outputs, but are detected by the error detection circuitry. These correspond to the faults in the original functional circuit that would have gone undetected without the presence of the error detection circuitry. The ratio of these detected faults to all faults that cause errors in the function outputs gives the coverage that is provided by the error detection

circuitry. This is the final coverage number that is reported by the coverage evaluation tool. If this coverage does not meet requirements, then the designer can use the CED insertion tool to insert a different CED scheme that will provide better coverage.

5. EXPERIMENTAL RESULTS

Experiments were performed using the above CAD tools on some synthesizable Verilog control logic designs. Designs 1, 2, and 3 were taken from the Torch processor. Design 4 was extracted from a Hewlett-Packard industrial design.

Table 1. Normalized Results for Designs

DESIGN 1				
Scheme	Area	Delay	Power	Coverage
No CED	100	100	100	0
Parity	109	101	140	75.3
One-Hot	112	101	145	77.0
Hybrid	190	152	165	93.0
Duplication	219	166	231	100

DESIGN 2				
Scheme	Area	Delay	Power	Coverage
No CED	100	100	100	0
Parity	114	126	106	79.1
One-Hot	130	171	115	82.2
Hybrid	220	215	240	94.0
Duplication	244	232	248	100

DESIGN 3				
Scheme	Area	Delay	Power	Coverage
No CED	100	100	100	0
Parity	140	130	120	79.4
One-Hot	170	190	133	85.0
Hybrid	215	150	230	94.1
Duplication	270	177	280	100

DESIGN 4				
Scheme	Area	Delay	Power	Coverage
No CED	100	100	100	0
Parity	118	160	150	78.4
One-Hot	145	185	151	76.7
Hybrid	190	183	190	92.3
Duplication	234	225	200	100

For each design, we used the CED insertion tool to automatically insert each of the four supported CED schemes: parity, one-hot, hybrid, and duplication. We then used a commercial synthesis tool with generic libraries to generate a gate-level implementation. We measured the coverage that was provided by the CED circuitry for each of the schemes using the coverage evaluation tool. We also compared the area, delay and power, for the designs with the CED circuitry versus the original designs with no CED. The results are shown in Table 1. Note that the values for the area, delay, and power, are normalized with respect to the original designs with no CED.

From the results, it can be seen that the 4 different schemes provide a broad range of cost/coverage tradeoffs that the designer can choose from. The parity scheme has the smallest area overhead in all cases, but also has the lowest coverage in all cases expect for Design 4 where it provided slightly better coverage than the one-hot scheme. In general, the one-hot scheme requires a little more overhead than parity and provides a little better coverage (Design 4 is an exception). To boost the coverage higher, the hybrid scheme can be used. It provided over 92% coverage in all cases with less overhead than full duplication. If 100% coverage is required, then the duplication scheme can be used, although it has the largest overhead.

6. SUMMARY AND CONCLUSIONS

This paper presented four CED schemes that can be automatically inserted into synthesizable Verilog RTL. A CED insertion tool was described for modifying the Verilog source to add the error detection circuitry. A coverage evaluation tool was also described for measuring the fault coverage that the CED circuitry provides in the gate-level implementation after synthesis. Using these two tools, the designer can rapidly explore various design points to find the best cost/coverage tradeoff that meets requirements. The experimental results demonstrated that the four CED schemes provide a broad range of cost/coverage tradeoffs to choose from.

The methodology described here can be easily incorporated into the typical design flow used in industry and provides a quick solution for designers to use to satisfy coverage requirements. As soft errors rates continue to escalate with each new generation of VLSI technology, automated design techniques for inserting CED circuitry will become very important.

7. ACKNOWLEDGEMENTS

The authors would like to acknowledge the contributions to this work by Mike Ziegler, David Fotland, Bill Bryg, Gary Gostin, Harry Foster and Lionel Bening from Hewlett-Packard and Sanjay Ramnath, Debaleena Das and Jayabrata Ghosh-Dastidar from University of Texas-Austin. This research was supported by a grant from the Hewlett-Packard Company.

8. REFERENCES

[Gorshe 96] Gorshe, S., and B. Bose, "A Self-Checking ALU Design with Efficient Codes," *Proc. of VLSI Test Symposium*, pp. 157-161, 1996

[Lo 92] Lo, J.-C., S. Thanawastein, and M. Nicolaidis, "An SFS Berger Check Prediction ALU and Its Application to Self-Checking Processor Designs," *IEEE Trans. on Computer Aided-Design*, Vol. 11, No. 4, pp. 525-540, Apr. 1992.

[Mak 82] Mak, G.P., J.A. Abraham, and E.S. Davidson, "The Design of PLAs with Concurrent Error Detection," *Proc. FTCS*, pp. 303-310, Jun. 1982.

[Nicolaidis 89] Nicolaidis, M., "Self-Exercising Checkers for Unified Built-In Self-Test (UBIST)," *IEEE Trans. on Computer Aided-Design*, Vol. 8, No. 3, pp. 203-218, Mar. 1989.

[Pradhan 86] Pradhan, D.K., *Fault Tolerant Computing: Theory and Techniques*, Vol. 1, Englewood Cliffs, NJ: Prentice-Hall, 1986, Chap. 5

[Touba 97] Touba, N.A., and E.J. McCluskey, "Logic Synthesis of Multilevel Circuits with Concurrent Error Detection", *IEEE Transactions on Computer-Aided Design*, Vol. 16, No. 7, pp. 783-789, Jul. 1997.