

VHDL/VITAL Fault Simulation

Dr. Paul Kilty, VEDA Design Automation

Abstract

This paper begins by analyzing current market developments to discover why fault simulation is enjoying a renaissance. Having established why fault simulation software is still needed, it examines why traditional fault simulation algorithms are not ideal for VITAL netlists, and outlines new algorithms, based on set theory, which give accelerated performance for VITAL based fault simulation. Finally, the paper suggests that extensions be considered to the VITAL standard so that faults may be modeled such as stuck-open, bridging, and delay faults for CMOS circuits.

Market Pressures

One axiom that has stood the test of time in EDA is the famous 'rule-of-tens'. The 'rule-of-tens' states the cost of finding and fixing a defect increases ten-fold with every stage in the process. An error at design simulation stage costs extra man-hours, worth thousands of dollars. A defect detected after manufacture incurs NRE charges of between 50 and 100 thousand dollars. If a defective product is released, the cost resulting from loss of customer goodwill is incalculable. This means that it is essential for designs to be testable, so defective parts are identified early in the design cycle.

One way of improving design testability is to use software for automatic insertion of scan chains. Industry experts who favor full scan design often declare that this practice eliminates the need for fault simulation. Nevertheless, Dataquest continues to forecast a steady 5% per annum growth in fault simulation. This is because automatic insertion of scan chains incurs significant area and performance penalties. For over 20% of ASIC designs, this additional overhead of full-scan is unacceptable, and this percentage is increasing.

The class of designs where fault simulation is superior to scan include :

- High-reliability, high volume applications for the automotive industry. Here the area overhead associated with scan insertion (5% or more) is not acceptable. Applications are ABS, airbag deployment, and electronic suspensions.
- High-reliability, low power applications for satellites, mobile phones and lap-top computers. The additional power consumed by scan chains is undesirable and frequently special timing techniques such as gated clocks or asynchronous design are employed to reduce power consumption still further.
- Specialist timing requirements. The addition of full or partial scan into a design may seriously degrade performance on a critical path timing and gated clocks to reduce power consumption.

The test strategy employed in these cases is to start with a test set of functional test vectors, optionally augmenting these with a mixture of automatically and manually generated test vectors. A fault simulator is required to calculate the fault coverage of test set under development, and to ensure that the final test set provides sufficient fault coverage.

VHDL is becoming the high level design language of choice. And now that VITAL compliant sign-off quality libraries are available, engineers can use VHDL throughout their design flow, from system level specification right through to design sign-off. This means that the same language can be used in all the phases of the design process for circuit descriptions and test-benches.

Now that VITAL is used for design sign-off, it is natural for users to demand VITAL fault simulation. Traditional fault simulation tools use a variety of non-VHDL technologies. Mixing languages and methodologies carries a significant risk of design inaccuracy, lack of testability and failure, so a VITAL compliant fault simulation solution is to be preferred.

Algorithms for VITAL Fault Simulation

As designers move away from test synthesis, they revert to the older technique of developing their own test vectors with support from fault simulation software. But designs are now specified in VHDL or Verilog rather than using schematic capture. Traditional fault simulators cannot be used as the semantics of the VITAL specification means that new fault simulation algorithms are required.

Traditional fault simulators are based on three algorithms, the parallel fault simulation algorithm, the deductive fault simulation algorithm or the concurrent fault simulation algorithm. These algorithms differ in the way in which they store, organize and propagate fault values. A description of all these algorithms is to be found in Abramovic [1].

The parallel fault simulation technique represents a particular fault at a particular location in the circuit as an individual bit in a machine word or set of machine words. Faults are propagated by performing group operations on entire words. Operations on words are basic functions of a computer, so this aspect of the calculation is rapid. However, the fault propagation calculations are carried out regardless of whether the faulty circuit differs from the fault-free circuit, so many unnecessary calculations are carried out. Overall, the parallel fault simulation technique gives compact and predictable memory utilization, but may be relatively slow, depending on the implementation used, especially for multivalued logic.

The deductive fault algorithm [2] simulates only the fault free version of the circuit and uses this to deduce which faults can then be detected at an internal node or primary output of the circuit. Overall, this technique is relatively fast, but is really only suitable for binary or three-valued logic.

The concurrent fault algorithm [3] holds fault values in a linked list data structure. A fault cell consists of a fault value, a fault number and a pointer to the next active fault. Overall, this technique is relatively fast, as only active faults (those where the faulty value is different from the fault-free value) are simulated, but the linked list structure means that, like the deductive fault algorithm, it requires large amounts of dynamically allocated memory.

Advanced fault simulators optimize for performance and memory utilization by combining these techniques. For example, HIFault combines the parallel and concurrent algorithms to represent data as a linked list, not of individual faults, but of groups of faults. Fault propagation is performed only on a group if it contains a fault value which differs from the fault free value. This is known as the Parallel Value List algorithm [4] and is used for gate evaluations. For functional constructs HIFault uses a deductive like algorithm which allows mixed level fault simulations. However, even these sophisticated techniques do not give the performance needed for VITAL sign-off simulation.

Central to the problem is the way in which VITAL cells are modeled using truth tables. Here we have an example of a VITAL truth table for a d-type flip-flop.

```
pres clk d q
  0 - - 1 ...row 0
  1 / 0 0 ...row 1
  1 / 1 1 ...row 2
  1 0 - S ...row 3
  1 S - S ...row 4
```

Which is equivalent to the following pseudo-code.

```
if (first row matches)
  assign right-hand outputs
else if (second row matches)
  assign right-hand outputs
etc...(for remaining rows)
else
  assign X's to right-hand outputs
```

Note that in VITAL (as in Verilog) if the input to the model does not correspond to any of the possible inputs described in the preceding rows, then all the outputs will be set to 'X'. That is the output defaults to X. This is why rows 3 and 4 of the truth table are necessary, as in these cases the output is unchanged. Note that both rows are necessary, and that they are not mutually exclusive. These semantics, incidentally, are the worst possible for the model developer. Of the time taken to write most models, at least 50% is spent on X handling, and most bugs occur in this area. For example it is not immediately obvious that both rows 3 and 4 are necessary. If, instead, the output defaulted to no change, and automatic X-handling is used, the device is completely described by rows 0 to 2 only.

When simulating a fault-free model, the simulator evaluates each line until the matching line in the truth table is reached, then evaluation of the truth table stops. But when performing a fault simulation, every line of the truth table must be evaluated as multiple versions of the faulty circuit are evaluated in parallel. Each version of the circuit, corresponding to a particular fault, may have a different set of input values from the fault free circuit, and may follow a different row of the table.

The solution is to define the multiple versions of faulty circuits as sets of circuits. So the pseudo-code for the VITAL truth table becomes

```
for the set of circuits which match first row
  assign right-hand outputs
for the remainder, for the set of circuits which match second row
  assign right-hand outputs
etc...
```

for the remainder
assign right-hand outputs to X

This is a deductive like algorithm, but maintaining separate sets for each possible logic value on a port or signal.

Developing this concept further, a d-type flip-flop with preset modeled using this approach looks like :

```
if ( row0_set = fault_get_set(cell, pres, L_0) ) {
    fault_output_set(cell, q, L_1, row0_set);
}

pres1_set = fault_get_set(ceil, pres, L_1);
cpR_set = set_intersect(fault_X01_value_change(cell, cp, L_0, L_1), pres1_set);
cpA_set = fault_active_port(cell, cp);
cpS_set = set_subtract_into(set_all(), cpA_set);

if ( row1_set = set_intersect(cpR_set, fault_get_set(cell, d, L_0)) ) {
    fault_output_set(cell, q, L_0, row1_set);
}

if ( row2_set = set_intersect(cpR_set, fault_get_set(cell, d, L_1)) ) {
    fault_output_set(cell, q, L_1, row2_set);
}

if ( row3_set = set_intersect(pres1_set, fault_get_set(cell, cp, L_0)) ) {
}

if( row4_set = set_intersect(pres1_set, cpS_set) ) {
}

if ( X_set = set_subtract5_into( set_union(cpA_set, fault_active_port(cell, pres)),
    row0_set, row1_set, row2_set, row3_set, row4_set ) ) {
    fault_output_set(cell, q, L_X, X_set);
}
```

It is important to replicate the true semantics of the table. The table has a row order precedence which is obvious from the pseudo-code for the logic simulation. Furthermore the rows are not mutually exclusive. It is essential that set subtractions are performed wherever the output sets are not mutually exclusive, always subtracting previous row output sets. Obviously the subtraction is redundant for the unchanging sets.

If the signal values are stored as a list of values, as in the parallel value or concurrent fault algorithms, the algorithm must diverge the list of values into the corresponding sets repeatedly at each evaluation. Continually diverging the list of values into sets, performing the set calculation and then reassembling the list is not efficient. The efficient alternative is to adopt the set algorithm. In the set algorithm, values of signals are stored not as a list of values, but as a group of sets, for each of which the signal has a specific value. A prototype implementation of a fault simulator using this algorithm gives very promising performance improvements.

But note that the VITAL semantics of defaulting to X without automatic X-handling necessitates the extra set operations to calculate the unchanging sets. For sequential

models these sets are often large. Whereas the explicit calculation of the X-sets themselves with automatic X-handling usually means dealing with much smaller sets.

Thus for fault simulation, Vital truth table semantics have a significantly poor effect on performance.

Representing the signal value as a group of sets leads to a different way of evaluating simple primitives such as xor gates.

example: xor

fault-model:

```
a_0 = fault_get_set(cell, a, L_0);
a_1 = fault_get_set(cell, a, L_1);
b_0 = fault_get_set(cell, b, L_0);
b_1 = fault_get_set(cell, b, L_1);

if ( set_0 = set_intersect( a_0, b_0 ) ) {
    fault_output_set(cell, z, L_0, set_0);
}
if ( set_1 = set_intersect( a_0, b_1 ) ) {
    fault_output_set(cell, z, L_1, set_1);
}
if ( set_2 = set_intersect( a_1, b_0 ) ) {
    fault_output_set(cell, z, L_1, set_2);
}
if ( set_3 = set_intersect( a_1, b_1 ) ) {
    fault_output_set(cell, z, L_0, set_3);
}
if ( set_X = set_subtract4_into( set_all(), set_0, set_1, set_2, set_3 ) ) {
    fault_output_set(cell, z, L_X, set_X);
}
```

For performance reasons it is important to minimize the number of sets of values that are represented. VITAL has a three valued input set {X, 0, 1} and a multi-valued output set.

Extending the VITAL standard to Fault Modeling

In some market areas there is increasing concern about defect levels and testing strategies for CMOS IC's. The SAF coverage is recognized as a poor metric. In the landmark paper by Hawkins et al [5] it is shown that CMOS test strategies based on defect classes will lead to lower defect levels. IDDQ tests and Boolean functional tests are shown to be complementary strategies addressing different defect classes. Neither IDDQ nor Boolean functional testing by themselves cover all defect classes. In general Boolean testing gives better coverage for open faults, but only partial cover for bridging faults, whereas IDDQ testing gives complete coverage for low resistance bridging faults and partial cover for open faults. A critical factor in the test strategy for bridge faults is the bridge defect resistance. For high bridge defect resistance, Boolean testing gives good coverage. The actual types of defects in CMOS depends on design, layout, and process technology, and will vary from foundry to foundry.

However many companies see IDDQ as an immature technology, complaining that testers are difficult to set up in practice. Some are requesting better fault models for CMOS instead.

Furthermore in some European design houses the test strategy is incorporated at the design level. Functional tests take into account the cyclic nature of testers; the tests are compacted to form the basis of test waveforms, starting with a mandatory 100% toggle coverage. Designers with an intimate knowledge of the functionality of smaller designs (up to 75K depending on complexity) are requesting fault simulation tools to facilitate incremental waveform development for high fault coverage. Their knowledge of controllability and observability issues for their design is of course crucial. They are requesting interactive tools which highlight in particular any controllability or observability problems.

The fact that designers are concerning themselves with test waveforms means that they are not afraid to consider more sophisticated fault models. CMOS fault models can be described in VHDL. For example a stuck-open faults on the X input drain for the assignment

```
Z <= X nor Y;
```

can be modeled as the transition fault

```
Z <= (X and not Z) nor Y;
```

where on the Right-Hand-Side Z is the output value in the current VHDL delta cycle.

There is an opportunity for the VITAL committee to address these problems and give guidelines, working towards standards which lead to an improvement in quality.

References

- [1] M Abramovici, MA Breuer and AD Friedman, "Digital Testing and Testable Design", Computer Science Press, 1990
- [2] DB Armstrong, "A Deductive Method of Simulating Faults in Logic Circuits", IEE Transactions on Computers, Vol C-21, No. 5, pp464-471, May, 1972
- [3] EG Ulrich and TG Baker, "Concurrent Simulation of Nearly Identical Digital Networks", Computer, Vol. 7, No. 4, pp39-44, April 1974
- [4] PR Moorby, "Fault Simulation using Parallel Value Lists" International Conference on Computer-Aided Design, 1983
- [5] CF Hawkins, JM Soden, AW Righter and FJ Ferguson, "Defect Classes - An Overdue Paradigm for CMOS IC Testing" International Test Conference 1994