The “Cell-Aware” fault model does an excellent job of expressing

potential in-gate defects. Unfortunately, some circuits contain

too many potential cell-aware faults to test for with the limited

resources available to test engineers. To avoid detailed analog

modeling and layout analysis for cell-aware fault extraction (which is not the goal of this paper), we modeled cell-aware faults using

a methodology wherein a subset of all potential cell-aware faults was extracted through stuck-at ATPG for study.

We differentiate these faults from their

analog/layout based counterparts by deeming them “Cell-Aware-Type” faults.

To determine the most important faults to target, we extracted

the “mandatory conditions” for detection for each potential

fault in two test circuits, and we performed functional simulation to

determine how many times these mandatory conditions were met.

We then analyzed the correlation between the mandatory condition

counts for each fault, and the corresponding fault’s actual detection during goodstate

simulation. Our results include the discovery that mandatory

conditions are a good predictor of a fault’s relative importance

in a circuit whose intended function is known, provided that the potential

state space in sufficiently explored during functional operation.

I. INTRODUCTION

A primary concern in test is the limited resources available

to test engineers. These constraints severely limit the breadth of

IC test. Most circuits are only tested over a brief period of time

or with limited computational resources. Granted, if resources

were unlimited we would want to test for every possible

manufacturing defect. Unfortunately, resources are scarce in any

industry, and especially so in the circuit design world. Even

with unlimited resources, if an average-sized circuit were tested

for every possible fault, it could take longer than the known age

of the universe to finish. We would finally be able to present a

tested product, and then the universe would collapse on itself.

Because of these limitations, it becomes necessary to shrink the

test space to a reasonable size. The work detailed in this paper

discusses how cell-aware test-space minimizing decisions can

be made by using functional simulation.

Fault models are used as aids by test engineers to describe

possible defects in a circuit and to generate input patterns to test

for them. One of the more predominantly used fault models

is known as the “Stuck-At Fault Model.” In this model, wires are

modeled as being stuck at logic value-1 (shorted to **V**dd ) or

logic value-0 (shorted to ground ). One essential fact about

this model is that defects occur on the lines between gates,

not within them. For example, to represent a potential stuck-at

fault, the top wire in Figure 1 is stuck-at 0 and is therefore

modeled as being grounded.

This particular fault will cause major problems because the

output of the circuit evaluates to 0 without regard to any other

inputs. Note that, in this case, the defect is described and conceptualized

as a problem within the boundary between primary inputs and

the gate, rather than occurring within the gate itself.

To test for any fault you must first excite the fault and

then observe the output. In this case we would need to make

certain that the first PI (Primary Input) was set to 1 in order

to excite the fault. We would also need to ensure that the

fault will propagate to the output of the entire circuit. For this

particular fault, only one pattern will excite the fault and allow

propagation to the output (inputs = 1111 ). Generally an ATPG

(Automatic Test Pattern Generation) tool is used to create a set

of possible patterns that could detect the faults in the fault list. In this research

we used both stuck-at fault ATPG and ATPG on a UDFM (user defined fault model), by taking advantage of a built-in feature

of Mentor Graphic’s Tessent.

For the research in this paper, we consider a fault model

known as the “Cell-Aware Fault Model.” This fault model considers

the transistor arrays within a logic gate and is explained

in more detail with an example in section III. The focus of

this research is to determine a method for assigning an “importance measure” to each cell-aware

fault. In other words, because we cannot focus on

all potential cell-aware faults, how can we decide which ones

present the most functionally likely flaws during normal usage?

To make this determination, we propose using an attribute

of a cell-aware fault detection called “Mandatory Conditions” during

functional simulation to rank faults. Functional simulation will

be discussed in more detail in Section IV. he way that

we performed functional simulation will be discussed in Section V. Results and

analysis will be discussed in Section VI.

II. PREVIOUS WORK

The cell-aware fault model is central to this research. To

place it in context, we begin with a discussion of its history.

The cell-aware fault model and its testing framework were

defined by Hapke et al. [1], although previous work along the

same lines was done by Sharma et al. [2]. Hapke has done

further work with the model by creating a tutorial for its use

in industry [3], applying it to a 32-nm laptop processor in a

case study [4], applying it to automotive parts in another case

study [5], creating a new testing procedure for it [6], showing

industrial results [7], and extending it to allow for detection of

some internal small delay defects [8]. Fan Yang et al. compared

results from the use of both the cell-aware and small

delay defect fault model in [9].

The method by which we perform functional simulation

was first proposed as a way to determine fault coverage

information at ITC in 2011 by Shi et. al [10]. This work itself

was an extension of an earlier paper that described targeting

very difficult stuck-at faults [11]. Our research is different

from this previous work due to the use of the cell-aware fault

model and the non-exclusive targeting of only hard stuck-at faults. On the

contrary, in this research many faults may be easy to detect.

Previously we have shown that some cell-aware-type faults are

resistant to n-detect, especially when a test set is optimized

for multiple detections of stuck at faults with very few patterns [12]. In the same paper,

we discussed using cell-aware-type top-off patterns as a means to

prioritize the detetion of the most important cell-aware-type faults when testing resources are limited.

[12].

III. CELL –AWARE-TYPE FAULTS

In this research the primary fault model of concern is the

“cell-aware fault model.” To differentiate this from other types

of fault models, consider the locations at which faults can occur.

For instance, consider the example of a stuck at fault in Section

I. As was stated there, a stuck-at fault can only occur on the

interconnects between logic gates. Perhaps a more technically

correct way of saying this is: defects can only be modeled as

occurring on logic nets. As another example, examine Figure

2, which represents two potential faults in the bridging fault model.

Notice again that the faults in this example occur on

electrical nets outside of the logic cells. To extend the analogy,

these types of fault models could be labeled as “cell-unaware

faults.” This is because the defects are all modeled without

respect to the internals of each of the standard cells in the

circuit. This is not to say that faults can only occur outside

the gates. Many faults occur within logic cells, but are able

to be effectively modeled (or at least detected) by fault models which represent

faults outside standard cells. In other words, even though the

manufacturing defect doesn’t produce a circuit exactly like the

one seen in Figures 1 and 2, these faults can be effectively

modeled as and tested for by thinking of the circuit as shown.

Unfortunately, the disregard for the internals of a standard

cell produce a large class of faults that cannot be deterministically tested.

Or rather, they will not be targeted during ATPG that uses a fault model

that only considers the electrical nets between gates.

Of course, some cell-aware faults may be guaranteed to be detected

by a test set that achieves 100% coverage of a different type of targeted fault. It is those

that would need to be detected fortuitously with which we are primarily

concerned.

To get

a better feel for the cell-aware-type faults we need to model, we will perform cell-aware-type

ATPG for the NOR gate in Figure 3. To discover

the internal faults that will not be tested for by a traditional

fault model, we will first perform ATPG with regards to the

traditional model. For this example (and in this research) we

use the stuck-at fault model.

There are six potential stuck-at faults that we need to test

for: *a* stuck-at 0, *a* stuck-at 1, *b* stuck-at 0, *b* stuck-at 1, *c*

stuck-at 0, and *c* stuck-at 1. To derive the stuck at test set, let

us find the smallest set of patterns that will test for all of these faults.

To test for a fault, we must first excite the fault, and

then we must observe the net at the output of the circuit. (Conceptually, observation can be exhaustively described

with the use of the Boolean derivative). To excite a fault, we

force a net to the opposite value than what it is stuck at.

To minimize the test set, the ATPG tool would then na¨ıvely

suggest:

With this test set, we can effectively test for all stuck-at

faults in our small example circuit. But we are simply modeling

faults as occurring on lines outside of logic gates. What if the

real fault is inside the circuit and can only be detected with

the test pattern ab (or a = 1 and b =1)? Figure 4 illustrates a

cell aware fault inside a nor gate, and describes the test pattern

that detects it.

A good understanding of the cell-aware fault model is

essential for understanding parts of our methodology. This

example is a good introduction to cell-aware faults. Ron Press

posted an excellent tutorial explaining these faults online in

2012 [13].

IV. FUNCTIONAL SIMULATION

Circuit simulation can be performed at different levels

of abstraction. Behavioral simulation is the highest level of

abstraction and functional simulation is the next highest. Technically,

the only difference between behavioral and functional

simulation is the inclusion of unit-delays within the circuit for

functional. The addition of unit delays for cells in the simulated

circuit allows for the comparison of “theoretical delays” among

different circuits. Behavioral or functional simulation is used

to used in industry to verify that a circuit is meeting its

intended function. During functional simulation, functional test

patterns are supplied to a circuit, and circuit output responses

are monitored to ensure that the circuit is functioning as

intended. This can be done in several different ways, but the

goal is to supply meaningful inputs to the circuit, and observe

meaningful outputs.

Consider any generic processor. If the processor receives an

instruction that does not contain a valid op-code, it will be put

into an unknown op-code state. In order to test this processor

for defects, we would like to supply it with instructions that

test all of the internal circuitry. Supplying it with one unknown

opcode would test certain parts of the exception handling

routines, but we shouldn’t supply the processor with only

instructions that have unknown opcodes. If we did, we would

not be able to effectively test the rest of the circuitry. Supplying

a circuit with many different inputs to test all of the corner

cases of the state space is the goal of proper functional

verification.

Consider Figure 5. This diagram represents the state space

of a fabricated circuit. We call S the set of all possible “Circuit

goodstates.” The size of the set S is equivalent to **2**m, where m

is the number of memory elements the circuit. Assume there

are three known functions of the circuit: A;B; and C . These

functions are known to use subsets of the states in S , denoted

by SA; SB; and SC respectively. During manufacture a defect

was introduced to the circuit. This defect can only be detected

if the circuit is operating in a state in goodstate grouping

denoted F.

Fortunately, this fault does not conflict with any of the

three known functions state spaces. Imagine that a customer

was only going to use functions A;B; and C . They would

never experience the fault, because they never operate the

circuit in a state space that overlaps with it. This means that

there would be absolutely no reason to discard this circuit,

because it would still be useful to the customer. By performing

this sort of functional simulation of circuits, we can rate

faults in terms of how many times they might be encountered

during functional use. Using functional simulation as well

as a few additional parameters to rate these faults in terms

of their general importance allows more efficient resource

allocation. This resource allocation is the primary purpose of

this research.

V. RESEARCH METHODOLOGY & EXPERIMENT

1. Cell-Aware Type UDFM Generation

In Section III several types of faults were discussed, but

in particular detail the cell-aware-type fault was described. We

also showed how to find a potential input vector that could

be used to identify a cell-aware-type fault that we should consider because it was not guaranteed to be detected by standard stuck-at-fatul ATPG (by a nor-gate example). The input vector identified for the cell-aware-type fault detection was illustrated in Figure 4. Note that this

represented only a single gate, and not a gate within a circuit.

The two circuits on which this analysis was performed are

an ISCAS‘89 benchmark circuit (s9234) and a DES56 encryption/

decryption circuit from opencores.org. Each of them was synthesized

using a different standard cell-library. These libraries included

many different types of cells, such as And-Or-Inverts, Half-

Adders, and Three-Input NORs. It became our task to develop

a standard way of finding which patterns could be used to

detect the cell-aware-type faults of interest. We did this in a very similar manner to

the example in Section III. To automate the process, we used

a standard stuck-at ATPG tool (Mentor Graphics Tessent) to

generate the stuck-at test patterns so we didn’t have to do

it manually (as was shown in the example). We then cross

-referenced each standard cell’s truth table and determined which

input patterns could be needed to detect a fault within that cell

and yet may not be covered by stuck-at fault ATPG. A subset

of these patterns were chosen as representing the detection conditions for our cell-aware-type faults.

The identified input patterns were then used to create cell-aware-type fault list Using the

UDFM (User-Defined-Fault-Model) functionality in Mentor Graphics Tessent.

After generating a UDFM

that properly represented several potential cell-aware-type fault

input-output pairings, we moved on to the next part of the

experiment.

1. Mandatory Condition Extraction

To allow for the prediction of the importance of specific

cell-aware-type faults in a given circuit, we first had to extract

an attribute of each fault individually. As was shown before in

Section III, certain input vectors are required to test for a given

fault. For the cell-aware-type fault in Figure 4, only one such input

vector was specified. For larger more realistic circuits that consist

of many gates, there are generally many different patterns that can detect

the same fault. However, there may still be certain characteristics of those

test patterns that must always be met for a fault to be detected. We

intend to use meeting those characteristics during functional simulation

as an indicator of whether fault detection may have been possible in functional

mode. However, to do this, we must first identify the relevant characteristics.

A certain ATPG approach used to enhance the probability of fortuitous detection of unmodeled defects is called “n-detect.” This approach

allows the user to specify a value of n, which corresponds to the minimum number

of detections desired for each fault in the test set. Thus, the specified

value can be used as a target an ATPG tool to generate a set of

patterns that detect each fault at least n times. Specifying a

large value of n will return many patterns that detect the same

fault. A matrix showing which faults are detected by

each pattern is specified an output file format called a fault dictionary. The fault

dictionary is essentially just a cross-listing of patterns and the

respective faults they detect.

We used a similar method to n-detect

to create a test set that detected each cell-aware-type fault multiple times.

This allowed for the generation of a fault dictionary, which

we then examined to extract a general formula for the patterns

that could detect a given fault. In other words, we discovered which

input conditions must always be satisfied for those faults to be

detected. These formulas are henceforth

referred to as the “mandatory conditions” for fault detection.

Imagine a circuit C has a potential cell-aware fault f , six

primary inputs, labeled: p0 􀀀p5 , and 2 internal state elements

(D-Flip-Flops, Latches, etc...), labeled d0 􀀀 d1 from most

significant to least significant. C is large enough that four test

patterns exist that can detect f . Assume we set the n-value to

some arbitrary number greater than four. After running ATPG,

using the UDFM generated above, the following patterns are

described in the fault dictionary.

Notice that the values in red do not change at all in any of

the patterns, but that every other bit changes. Thus, for this fault, there are three

mandatory conditions. We denote the mandatory conditions of

f as MC**(**f**)** . We can now derive MC**(**f**)**  for this circuit,

because we know all of the patterns that detect the fault.

In general, it is difficult to know how many patterns are capable

of detecting a particular fault without exhaustive analysis. Thus,

it is important that the value of n selected be appropriately large

to help increase the probability that the identified conditions are truly mandatory.

The next part of the setup is to count how many times

these mandatory conditions are met during functional usage.

However, we must first discuss how we performed functional

simulation.

1. Circuit Goodstate Extraction Functional Simulation

For s9234, we did not know the original function of

the circuit. We intentionally allowed its intended function to

remain obscured so that we would only test for a certain small part

of the overall state space. This experiment is representative of

rating the importance of cell-aware-type faults for a circuit whose

function is unknown, or only one function of a group of many

is going to be used. This was to emulate a company using an obfuscated

circuit for which they only know (or have) one intended use. We first

inserted a scan chain into the circuit in order to allow for

the extraction of state bits. We then generated over 40,000 random

inputs, and extracted the values of all of the state bits after

each clock cycle. We stored a set of 40,000

goodstates, and then generated another set. Finally we created

a test bench that allowed the fault simulation of s9234,

using the goodstates.

For DES56 however, the intended purpose was known. We

also had a functional verification test bench for this circuit already which

was supplied by the designers on opencores.org. We used the DES56 circuit

to encrypt and decrypt the plain text ASCII version of this

paper [14]. This qualifies as functional simulation of the circuit

because it is realizing the intended purpose of both encryption

and decryption. Due to both the nature of the circuit and

the inputs applied, we covered a much higher percentage of the

state space during this functional simulation. This was intended

to emulate the testing of a circuit whose functionality is known

and will be used in its entirety.

.

Lastly, we detected the number of times that a fault’s

mandatory conditions were met during functional simulation.

This was the key attribute that we intended to use for fault

importance predictions. We henceforth refer to this value as

the “mandatory count” for a particular fault. This is discussed

in the next subsection.

1. Mandatory Counts during Functional Simulation

As we have seen, the mandatory conditions for a given

fault can be represented as a Boolean function of the fault.

In the example above, we determined that MC**(**f**) =** p**0**p**5**d**1** .

The result that these mandatory conditions can be represented

as a Boolean function is essential. By representing them as a

boolean function, we can add a few simple logic gates with

an output signal that feeds a counter. We created a script to

read the mandatory conditions file that was created by parsing

the fault dictionary. This script output a testbench to allow

for functional simulation of the circuit, but it added a few

of these extra logic gates that allowed us to keep a count of

how many times the mandatory conditions for each fault were

met during functional simulation. The logic gate that would be

added to the circuit’s test bench in order to examine whether

the mandatory conditions for the fault detailed above is given

in Figure 6.

Fig. 6. Mandatory Condition Detector for fault f

Because the mandatory conditions were generated based on

test patterns that were engineered to propagate the fault to the

outputs of the circuit, if they are met during functional simulation,

then a device with this fault will output an erroneous

value during its use. We can use the mandatory counts for

each fault to rank them in terms of their importance. They

can then be sorted by the probability that they will cause

problems during the functional use of the device. In order to

determine if the mandatory counts were a good indication of

whether or not the fault would be detected during functional

use, the last part of the experiment consisted of the use of

a fault simulator to determine whether or not a fault would

be detected during the same “goodstates”. We could then

set a prediction criteria in terms of the mandatory counts

for a particular circuit, construct a confusion matrix, and

calculate the corresponding statistics to determine how well

these mandatory counts predicted whether or not a fault was

truly functional (would be detected during fault simulation).

VI. RESULTS & ANALYSIS

To start out, we used a very basic selection criteria for

predictions and observations. If a fault’s mandatory count

was greater than zero, then the fault was “predicted” to be

a functional fault. Similarly, if a fault had any goodstate

detections during simulation, then the fault was “observed”

to be a functional fault. In future efforts we will refine this

criteria and examine the resulting increases or decreases in

precision.

A. Mandatory Counts as Fault Classifiers

As mentioned in the preceding section, we have analyzed

two particular circuits during this research. The first of which

was an ISCAS’89 benchmark circuit called “s9234”. The

circuit contains sequential logic, and 9,234 nets were a stuck-at fault may be present. Other

than those two facts, we did not determine the true function

of the circuit. This allowed us to emulate a testing procedure

when the function of a circuit is unknown. (Alternatively,

it could represent the case where only a subset of all possible functionality

is intended to be used.)

After performing

the experiment described above we determined that there were

1,034 detectable cell-aware-type faults from our fault list in the circuit. We claim that

a fault is predicted to be functional if the mandatory count

for that fault is greater than 0. In other words, mandatory

conditions predict a fault to be functional if they are detected

at least once during functional simulation. Similarly, we define

the observance of a functional fault as occurring if the fault was detected

one or more times during goodstate fault simulation. Therefore,

we can construct the confusion matrix in Figure 7 to describe

the usage of mandatory conditions as a predictor when the

function of a particular circuit is unknown, and is guessed by

randomly stimulating the inputs during simulation.

We also calculate several statistics in Figure 8 to determine

how well mandatory counts predict in this case.

In summary, mandatory conditions are really not a spectacular

indicator of whether or not a fault is functional when

the function of the circuit is unknown and/or when the state space

is not sufficiently traversed during functional simulation (possibly because

only part of the space is intended to be used).

In this circuit, multiple flip-flop values had highly skewed probabilities of

being equal to a logic one or a logic zero. This meant that it was

very difficult to fortuitously meet any of the set of “non-mandatory” conditions

for a cell-aware-type fault in some cases—even when the mandatory conditions

themselves were met many times. This was especially true when

detection of a cell-aware-type fault required many care-bits in its test patterns, but when there

were also multiple alternative ways of realizing those care-bit assignments. This led to few

mandatory conditions that were common across all detection patterns, and high mandatory

condition detection counts.

In future efforts

we will manipulate the prediction criteria more to determine

if increasing the required mandatory count before a fault is

“predicted” functional will have a positive effect on these

statistics. With the second circuit we tested however (DES56)

the function was known, and we had much better results. The

confusion matrix for this circuit is in Figure 9.

Again the statistics were calculated based on the matrix

(Figure 10), and this time we saw much better results.

We can see that this is an extremely accurate classification

methodology if you know the functionality of the circuit and if the

state space is sufficiently traversed in functional mode. We

can calculate the **F**1 score of the predictor as well (values

closer to one represent a better classification scheme). The

**F**1 score for this confusion matrix is 0.997835. This is a very

high **F**1 score, which again corroborates that mandatory counts

during functional simulation can be excellent classifiers. The final

statistic produced in regards to this confusion matrix is known

as Matthew’s correlation coefficient. This is a good segue into

the next analysis of our data, as it represents only a single

way that the correlation of two variables can be calculated,

but specifically regards the false negative and false positive

rates described by the confusion matrix. Matthew’s correlation

coefficient is calculated to be 0.93339. Again this is a very

high value, which indicates that these mandatory counts were a

spectacular predictor of functional faults in this experiment.

In the future we intend to extend this analysis by increasing

the mandatory count before a particular fault is predicted, and

drawing an ROC to analyze if there is any increase in precision.

1. Regression Analysis

In addition to the standard confusion matrix analysis for

a predictor that was done above, we also determined the line

of best fit for our data, and calculated Pearson’s correlation

coefficient and the **R**2 value for our data. Somewhat surprisingly,

there was a high correlation for the first circuit, when

using Pearson’s correlation coefficient. It was determined that

**\_** = 0**:**77, which indicates that there is a strongly positive

correlation within the data. This became slightly more obvious

when a plot of the data as well as a linear regression line was

produced. This plot is shown in Figure 11.

The regression parameters for this model are: slope= 0.55

Intercept= -90. We can calculate the R-squared value by

squaring **\_**  from above, R-squared = 0.5929. As was seen

in the confusion matrix analysis, these values are strongly

correlated, but not quite as strongly as we had hoped to see.

A takeaway from this is that while we can’t really trust

particular detections for circuit whose functions are unknown,

we can sometimes predict about how important certain faults are relative

to one another. We can make this extrapolation because the

regression analysis of this section is concentrated on the actual

mandatory counts, and goodstate observations. As opposed to

the confusion matrix analysis, which was only concerned about

the presence of mandatory counts and goodstate observations.

Although we have not proven it to be the case, it is likely that the faults

with the poorest predictions correspond to faults with few values that

have mandatory conditions and yet who have non-mandatory conditions that correspond

to flip-flops with highly skewed values during functional simulation.

A similar analysis was performed on the DES56 circuit,

and we again had more promising results. Pearson’s correlation

coefficient was calculated, **\_** = 0.88 The linear regression

model was plotted (Figure 12).

The regression parameters were calculated as: slope =

0.719, intercept = -116. This time, we found a much higher

R-squared value, which was 0.69. Again, we can see that when

the functionality of a circuit is known and when the state space

was more fully traversed (i.e. flip-flop values were not highly skewed), mandatory counts are

a much better indicator of general fault importance.

VII. CONCLUSION

The cell-aware fault model is excellent at describing faults

that could occur within standard cells. However, there is a very

large number of potential cell-aware faults in any given circuit.

To more efficiently allocate testing resources during

test, functional simulation with mandatory condition counters

can be performed beforehand. This allows the tests engineers

to more effectively target only those faults that are the most

likely to cause errors during functional use.

Using mandatory conditions for functional fault prediction

has a wide variety of applications. We presented the proper

background material and detailed our procedure

to make it easily repeatable. We provided examples

of mandatory condition calculations, and showed how they

could be used to predict whether or not a cell-aware fault is

functional. Future work will explore additional metrics

to better predict fault detections—especially in cases where a fault

has few mandatory conditions on its input variables, and yet

the number of care bits in its test patterns is large.

The results of

this paper also have some potential applications in intelligent

speed-binning techniques. To elaborate, one could test for

wider and wider scopes of functionality, and speed-bin circuits

more intelligently—especially if timing, paths, and multiple clock cycles

were added to the overall analysis. There are also potential applications of

this methodology to artificially intelligent testing systems.

The

results of this research are applicable to many different testing

methodologies, and should not be restrictively thought of as

purely cell-aware specific. There is still much work to do in this

area. We will likely proceed to test more circuits and will perform

similar analysis. This will allow us to pinpoint more precisely

the kinds of circuits for which these techniques are most effective.