Delay Fault Testing of Integrated Circuits *

Alonna Albertus
afalbertus@cae.wisc.edu

Jacob G. Hansen
jacob@pbk.dk

Martin Hans
mhans@gmx.net

Michael Kristensen
c973396@student.dtu.dk

Abstract

In order to be able to give guarantees on the operating speed of a circuit, delay fault testing must be applied. This paper attempts to give an overview of the field. First, the path delay fault model is described and compared to the major competing delay fault model; the gate delay fault model. Then, deriving and minimizing test sets using the path delay fault model is explained, as well as the synthesis of circuits that are readily delay fault testable. Finally, the limitations of real-world chip testers and the consequences they have on circuit performance guarantees are discussed.

1. Introduction

To be sure a circuit functions properly, several tests must be performed to verify the correctness. A stuck-at test can be used for functional verification and Iddq testing can detect shorts between nets, which stuck-at tests are unable to. A third test method is the delay test, which is used for verification of the dynamic behavior of the circuit. With decreasing process sizes in modern deep sub-micron technologies and increased demand for speed, a small defect in the processing can result in a propagation delay outside the acceptable bounds. In a short path through the circuit an extra delay might not be critical, but if the delay of longer paths is increased, the overall speed will suffer from the delay fault.

2. Delay fault models

Two models are considered for modelling delay faults: the *gate delay fault model* and the *path delay fault model*. The gate delay fault model considers the inputs and outputs of a single gate and the path delay fault model considers the entire path from the inputs of the circuit through gates and nets to the outputs of the circuit.

Common to both fault models is the test method. First an initialization vector V_1 is applied at time t_1, then another vector V_2 is applied at t_2 such that the faults are sensitized. After all signals are expected to have stabilized, the output from the circuit is sampled at t_3. If a delay fault is present, the output sampled at t_3 might differ from the faultfree output.

2.1. Gate delay fault model

The gate delay fault model assumes all faults are located at the inputs or outputs of a gate in the circuit. Thus faults are modelled locally, and timing failures will only occur on a single gate.

In [3] a gate delay fault model is presented that models slow-to-rise and slow-to-fall transition faults. The model consists of the boolean functions AND, NAND, OR, NOR and NOT, a special FANOUT-gate for dealing with fanouts, and an IDENT-gate representing wire delays.

A transition on the output of a gate is represented by an initial value (IV) and final value (FV) as shown in Figure 1. The start of the transition is indicated by *earliest arrival* (EA) and the transition ends at *latest stabilization* (LS). During a transition the output of the gate is undefined.

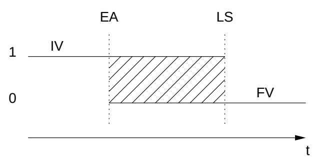


Figure 1. 1-to-0 transition on gate output

By describing the rising- and falling delays of gates with the nominal delay of the gate and a manufacturing tolerance, σ, four gate-characteristic parameters can be derived, see Figure 2.

With the four parameters, the values for EA and LS can be calculated for the faultfree gate. By adding a fault with size λ to the gate, the values for EA and LS will increase by the amount λ, as shown in Figure 3 [3].

*This paper is handed in as an assignment answer in lecture course *02208 Test of Integrated Circuits (2002)* at the Informatics and Mathematical Modelling department, Technical University of Denmark.

$DAC = D_{fall} - \sigma$	arrival (controlling)
$DAN = D_{rise} - \sigma$	arrival (noncontrolling)
$DSC = D_{rise} + \sigma$	stabilization (controlling)
$DSN = D_{fall} + \sigma$	stabilization (noncontrolling)

Figure 2. NAND-gate delay parameters

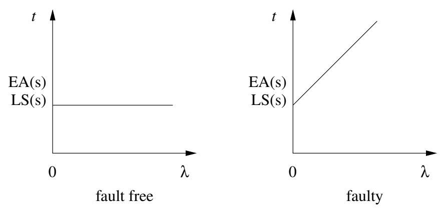


Figure 3. $\lambda's$ impact on EA and LS

To *definitely detect* a gate delay fault [3], EA must be greater than the observation time $Obs(z)$ at the primary output z, and a threshold ε must exists that satisfies

$$EA(z,\lambda) > Obs(z) \quad \text{for all } \lambda > \varepsilon$$

The equation states that a fault must have a certain size to be definitely detected. A weaker requirement, *possible detection* [3], can be used when no solution to the above equation can be found, though it requires further analysis of the fault.

2.2. Path delay fault model

Where the gate delay fault model models faults locally, the path delay fault model accumulates all the delays from the inputs to the outputs.

A path delay model is described in [4]. The delay of a connection e is denoted by $\delta^r(e)$ for a rising transition and $\delta^f(e)$ for a falling transition. In a similar way, the delay of the path P is $\delta^r(P)$ or $\delta^f(P)$. By accumulating all delays along a path P, and testing if the delay is within bounds $[\tau_{min}, \tau_{max}]$, the presence of a delay fault can be determined:

$$\tau^i_{min} \leq \sum_{e \in P_i} \delta(e) \leq \tau^i_{max}$$

2.3. Comparison of gate- and path delay fault models

As shown in the preceding sections, the gate delay fault model models a rising or falling transition on the output of a single gate. Modelling a delay fault on a gate, requires a rising and falling transition on all inputs and outputs of the gate. Thus the number of possible faults in the gate delay fault model increases linearly with the number of gates.

$$\text{faults}_{\text{gate}} = 2 \cdot (\text{inputs} + \text{outputs})$$

In contrast, the number of faults modelled by the path delay fault model has a worst-case exponential growth, depending on the specific circuit.

In the gate delay fault model, a delay fault on a gate might be "hidden" if some gate in the fault propagation path has a delay shorter than expected. Likewise a small delay fault in a gate can be impossible to detect, if the threshold ε is too small. As the entire path is tested in the path delay fault model, it doesn't suffer from these problems. In addition, sometimes delay faults on only part of the path will not affect the timing of the whole path. The path delay fault model more closely models the delay from inputs to outputs of a circuit, and therefore can better guarantee the temporal correctness of the circuit.

3. Principles of test pattern generation for path delay faults

The overall procedure to generate test patterns for path delay faults follows in three steps:

1. Sensitizing a path through the circuit to propagate a fault.
2. Applying test inputs to the sensitized path.
3. Collecting outputs and checking them against predefined correct results.

Reference [5] introduces a five-valued logic to model the faults. The values are:

S0	steady 0
S1	steady 1
U0	changing to 0
U1	changing to 1
XX	unknown (don't-care)

These values are connected logically as in Figure 4 where a signal transition can be described when moving from one state to the other in the tree. The values U0, U1, and XX are depicted in Figure 5. The five logic values can be put into implication tables, like that of Figure 6 for an OR gate.

By expressing the input values of the circuit in terms of the five-value logic, the results can be propagated through the circuit. The results on the outputs expressed in the five-valued logic can then be used to determine which delay faults the test vector pair detects. The choice of input values can either be done by random choice of inputs followed by comparing to a delay fault list or by deterministic methods.

The circuit in Figure 7 shows how input values have been chosen to test for path delay faults in the path $\alpha - \beta - \gamma - \lambda$

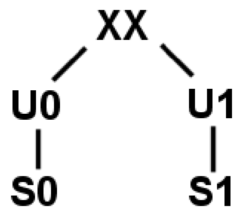


Figure 4. Logic value tree

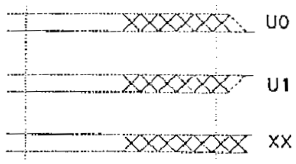


Figure 5. The U values

for falling value (U0). The values could have been chosen randomly by an algorithm or manually generated by methods comparable to those described for the gate delay model in [2].

3.1 Minimal test set

A robust test is a test that will find a delay fault independently of other faults in the circuit. Robust tests can be derived in different ways. Finding a robust test set for a circuit is feasible, but unfortunately the resultant test set is usually unnecessarily large. Minimizing the test set is much more time consuming. A complete path delay test must contain test patterns to test for all delay faults. As the number of paths rises exponentially with the number of gates, this method quickly becomes infeasible when the number of gates increases. It can be shown that 100% delay fault testability is not necessary to guarantee the speed of a combinational circuit [4]. The concept of delay-verifiable circuits will be discussed in Section 4 of this paper.

a\b	S0	U0	S1	U1	XX
S0	S0	U0	S1	U1	XX
U0	U0	U0	S1	U1	XX
S1	S1	S1	S1	S1	S1
U1	U1	U1	S1	U1	U1
XX	XX	XX	S1	U1	XX

OR-gate: c = a OR b

Figure 6. Implication table for an OR-gate

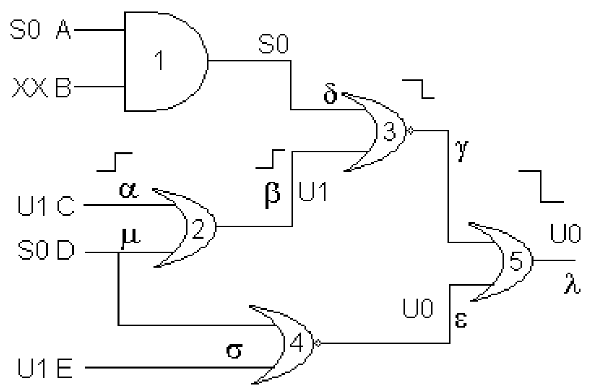


Figure 7. Sample circuit using five-valued logic (fig. 5, p. 697 in [5])

3.2. Optimizing test patterns

Path delays can be divided into two groups of delay faults according to whether or not the fault will be visible on the outputs. The first group of faults contains the *robust testable path delay faults* (RPDF), which are faults that can be detected using a sensitized path through the faulty path to the output. The second group are the *robust dependant faults* (RD). These faults are physically "hidden" in the design. One way this could happen is if the gates are fed by other non-faulty inputs that arrive later than the input delayed by the fault. Therefore the timing constraints of the gate will still hold. Another possible cause for the "hiding" of the delay fault is that the fault exists in a redundant area of the circuitry. In either case, the presence of RD faults will not affect the timing of the circuit. RD faults therefore have no significance to the circuit functionality unless other non-RD faults are affecting the circuit.

The distinction of these two groups of delay faults narrows the search of a complete path delay fault test set down to finding a complete test set for RPDFs, i.e. finding a complete path delay fault test set and disregarding the test patterns that solely test for RDs faults.

To sort out RD faults in a circuit C, the circuit is converted to a fanout free non-inverting (no inverters in the circuit except at the PIs) circuit C'. A pseudo-code algorithm has been developed in [4].

```
Convert C to an internally non-inverting
      circuit C' R = {}
While(C' has any gate with fanout>1) {
  While(C' has a redundant stuck-0
    connection e){
      Replace e by constant 0
      R = union of R and e
    }
    Find gate g with fanout > 1
    Duplicate g and move fanout
    - connections so each duplicate
    - has less fanout than g
}
All paths with some connection in
R form rising RD net.
```

The pseudo-code transforms the circuit to a non-inverting circuit by moving the inverters to the PIs and adjusts the functional operations of the gates by Boolean logic. Then all gates with fanout greater than 1 are duplicated until the fanout reaches 1 and each fanout connected to different inputs at following gates. By doing this the code can evaluate which gates are redundant and remove these by disconnecting their inputs and connecting a logic '0' instead. These redundant gates indicate that there is one or more redundant path(s) in the circuit and yields the RD set that does not need to be applied to the circuit to guarantee the functionality of the circuit.

It is noted in paper [4] that a stuck-at fault test would, with high probability, detect path delay faults when running the test at the same clock speed as the circuit is intended to run after fabrication. This is due to the fact that in stuck-at fault testing all nodes are sensitized and given values both '0' and '1'. Running at full clock speed will then detect major delays as stuck-at faults. Many of the path delay faults that are not detected in a stuck-at fault test may be RD faults. However, without specific delay fault testing there can be no guarantees made about the temporal functionality of the circuit. Delay fault testing can be very expensive, and so as usual there will be a trade-off between quality of test and cost of test.

3.3 Parallel path delay fault testing

Testing for faults in parallel can clearly optimize time-consuming test pattern runs. This can be done in several ways. Parallel testing can be performed when two tests are disjoint, i.e. they share no common inputs or common paths in order to propagate a test value to the POs. Furthermore semi-disjoint parallel testing can be performed by first noting (manually or by using an ATG algorithm such as the one described in [2]) which input values are used for propagating the test value (sensitizing) and which path propagates the test value to the POs. Then parallel testing can be performed if two 2-pattern tests <v1, v2> can be found where:

- All requirements to sensitizing values are compatible in the two 2-pattern tests and
- No two test values are propagated by the same connection.

Even using the same paths for propagating several test values can yield parallelism. In [5] (p. 697-698) it is noted that several faults can be tested using the knowledge of the timing of the circuits. The paper presents the circuit in 7 where a fault (slow to fall through path $\sigma - \varepsilon - \lambda$) is "hidden" by the more time consuming $\alpha - \beta - \gamma - \lambda$ path (falling). The robust test for the falling delay of the first path is therefore also a test for the falling delay of the second path. Should the delay on the second path exceed the delay in the first path, then the delay would be visible on the output. The delay fault is robust dependant (RD) when it holds that the delay is equal to or less than the difference D between the delays on the two paths ([4], definition 4.1). If the delay is greater than D, then the test will remain robust. Therefore, we do not need to explicitly test for this delay fault.

4. Synthesis of delay-verifiable circuits

As explained in Section 2, there are two models for delay faults: the gate delay model and the path delay model. This section further explores delay fault testing using the path delay model. Specifically, we aim to improve the testability of a combinational circuit by understanding the concept of delay-verifiable circuits. We will show that certain faults that have previously been defined as untestable can be verified to function correctly. First the terms will be defined and then an example will provide further understanding. Finally, methods are introduced to implement delay-verifiable circuits.

4.1. RV-testable circuits

In [6] and [7] the following terms are defined. A *robust test* will detect a particular path delay fault independent of delays in the rest of the circuit. If a test does not possess this property it is said to be a non-robust test. However, some non-robust test can be validated, but they depend on other paths being tested robustly. A *validatable non-robust (VNR) test* will detect a particular delay fault only if certain other paths pass robust tests. VNR tests can be added to a robust test set in order to increase the test coverage. If a circuit has a robust or VNR test for every path delay fault, then it is called an *RV-testable* circuit. RV-testable circuits require each output function to be prime and irredundant which may increase the area overhead of the circuit [6]. Furthermore, not all circuits are RV-testable. This creates a desire for a circuit to contain some non-RV-testable paths, but still be verifiable for temporal correctness.

4.2. Delay-verifiable circuits

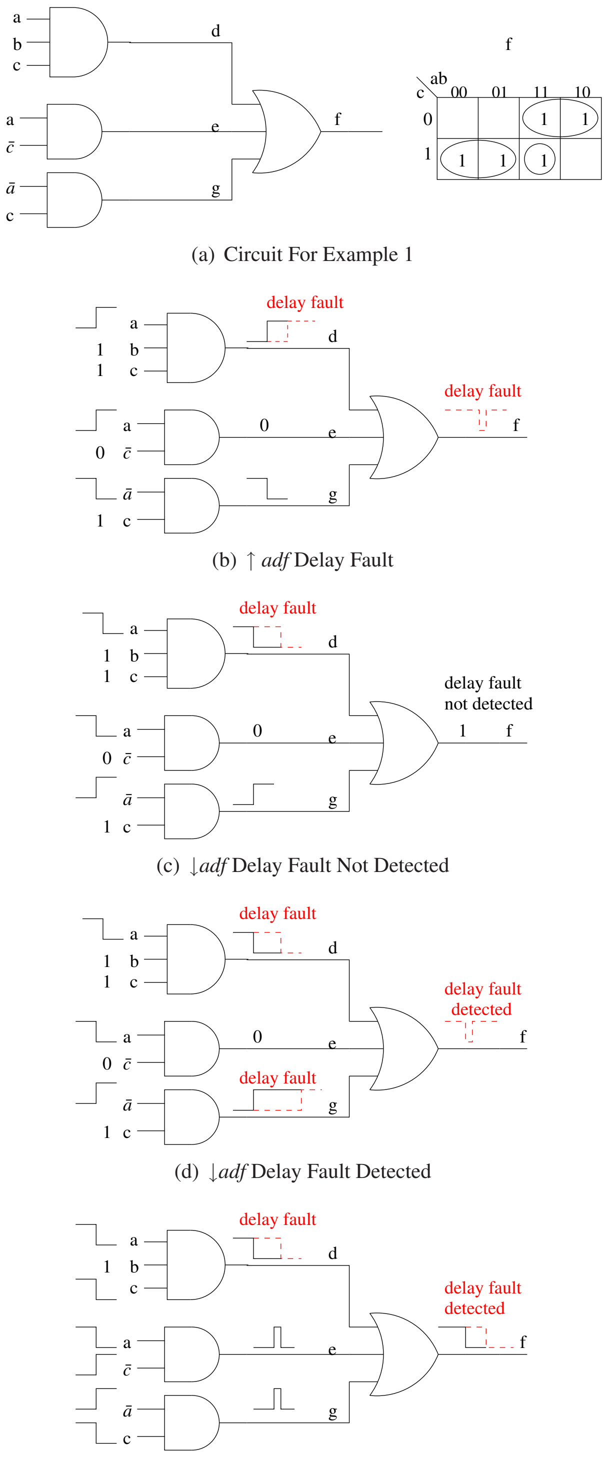
An approach has been presented in [6] that uses *delay-verification test sets* to verify the temporal correctness of a circuit without requiring every path to be RV-testable. In this approach, circuits are *delay-verified* if each path delay can be guaranteed to either a) fall within the specified delay limits or b) not affect the circuit timing. Unlike RV-testable circuits, delay-verified circuits do not have the requirement that the implementation of each output is prime and irredundant. Therefore, this approach will allow a more area-efficient implementation of two-level circuits, which has been shown by experimental evidence [6].

In addition to tests for single path delay faults, it may be necessary to test for multiple path delay faults. A *multiple-path-delay-fault (MPDF) test* is a test vector that includes multiple transitions on the inputs in order to test for a set of path delay faults. MPDF tests are similar to tests for single path delay faults, in that it can be robust or validatable non-robust.

The following examples taken from [6] will help the reader to further understand the concept of delay-verification test sets. Example 1 shows a non-RV-testable circuit that can be delay-verified.

Example 1: In Figure 8(a) we can see that the implicant abc is non-prime and we can conclude that the circuit is non-RV-testable. However, we would like to further examine the path delay faults to determine if a delay-verification test set can be found. In the circuit in Figure 8(a), it is easy to see that all path delay faults are detectable by robust tests, except those on paths adf and cdf. Delay faults on $\uparrow adf$ and $\uparrow cdf$, however, can be tested by VNR test sets. Fault $\uparrow adf$ depends on the falling delay of g in the path $\downarrow \bar{a}gf$, as shown in Figure 8(b). However, if path $\downarrow \bar{a}gf$ can be proven to be delay faultfree, then the fault will be detected. The VNR test vector $\{011, 111\}$ for this fault $\uparrow adf$ can be validated with a robust test $\{001, 101\}$ for $\downarrow \bar{a}gf$. Similarly, it can be shown that a delay fault on $\uparrow cdf$ can be detected with a VNR test vector $\{110, 111\}$ and validated by a robust test $\{100, 101\}$ for $\downarrow \bar{c}ef$.

The delay faults $\downarrow adf$ and $\downarrow cdf$ need further attention, because they are untestable faults. In other words, neither a robust test nor a VNR test exists. The untestable fault $\downarrow adf$ is illustrated in Figure 8(c). Any delay on the falling of d will be masked by the rising edge of g due to the path $\uparrow \bar{a}gf$. The same can be said about the fault on $\downarrow cdf$. However, if the circuit passes all other tests (robust and VNR) it can be



(a) Circuit For Example 1

(b) $\uparrow adf$ Delay Fault

(c) $\downarrow adf$ Delay Fault Not Detected

(d) $\downarrow adf$ Delay Fault Detected

(e) Multiple Path Delay Fault Test for $\downarrow adf$ or $\downarrow cdf$

Figure 8. Example 1: Delay-verifiable circuit

seen that the delays of $\downarrow adf$ and $\downarrow cdf$ will fall into one of these three categories:

1. Both delays are within the specified range and therefore the timing of the circuit will not be affected.

2. One of the two delays fall outside of the allowable limits, therefore causing a faulty path. However, the output f will be incorrect only if there is a corresponding late signal on another path. For the fault $\downarrow adf$ a delay on the rising edge of g could cause a glitch on the output f as shown in Figure 8(d). But, as long as g is not faulty, then the output f will operate correctly. The same argument can be made for the single delay fault on $\downarrow cdf$. Therefore, as long as the circuit passes all other delay fault tests (including the tests for $\uparrow \bar{a}gf$ and $\uparrow \bar{c}ef$) then a faulty path on either $\downarrow adf$ or $\downarrow cdf$ will not affect the circuit's timing.

3. Both delays are beyond the specified range, and therefore they both cause faulty paths. In this case, an MPDF test vector $\{111, 010\}$ will detect that both delay faults are present. This case is illustrated in Figure 8(e).

These are the only three possibilities for the delay faults on $\downarrow adf$ and $\downarrow cdf$. Therefore, we simply need to add the test vector $\{111, 010\}$ to guarantee temporal correctness.

This example shows that the circuit in Figure 8(a) is delay-verifiable and will operate correctly (temporally) as long as it passes the delay-verification test set, which must include a special test vector for a multiple path delay fault.

4.3. Implementation of delay-verifiable circuits

Upon understanding how a non-RV-testable circuit can be delay-verifiable, it is interesting to investigate the implementation of a delay-verifiable circuit. For an RV-testable circuit, one must implement each output to be prime and irredundant. This is not necessary for a delay-verifiable circuit. Article [6] suggests the following procedure for implementing a delay-verifiable circuit.

1. Multiple outputs in the circuit should be optimized to be prime and irredundant, using a tool such as ESPRESSO. Sub-circuits implementing each output function are required to be irredundant, but not necessarily prime [7].

2. If the circuit is not delay-verifiable, then the circuit must be modified. Non-prime literals in the sub-circuits of each output function can be removed until the circuit becomes delay-verifiable.

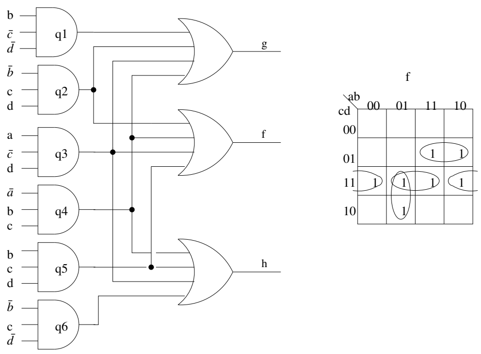
inputs				outputs		
a	b	c	d	g	h	f
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	0	1	1	1	0	1
0	1	0	0	1	0	0
0	1	0	1	0	0	0
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	0	0	0
1	0	0	1	1	1	1
1	0	1	0	0	1	0
1	0	1	1	1	0	1
1	1	0	0	1	0	0
1	1	0	1	1	1	1
1	1	1	0	0	0	0
1	1	1	1	0	1	1

Figure 9. Truth Table For Example 2

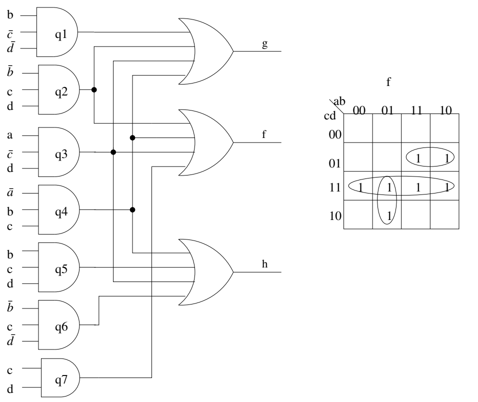
An example taken from [7] illustrates this implementation method of delay-verifiable circuits.

Example 2: The truth table in Figure 9 describes a multi-output combinational circuit. The function is minimized using ESPRESSO. Figure 10(a) shows the resultant circuit that requires six AND gates to implement the three output functions g, h, and f. All of the faults in this circuit are RV-testable except faults on $\bar{b}q_2f$ and bq_5f. These faults are non-RV-testable because they both include the literals c and d, and the literal b in both inverted and non-inverted form. Thus, the rising and falling transitions on gate q_2 will be masked by the transition on gate q_5 and vice versa. The procedure noted above requires us to remove non-prime literals until the circuit becomes delay-verifiable. In this case, either the non-prime literal $\bar{b}$ must be removed from gate q_2 or b from gate q_5. If b is removed from gate q_5, the term cd must be added to the output f. This corresponds to gate q_7 in Figure 10(b). Since the term bq_5f is still needed for the output h, it cannot be removed completely. However, by removing b from function f we have transformed the non RV-testable circuit into a delay-verifiable circuit. The new circuit shown in Figure 10(b) contains seven product terms. In [7] it is shown that when each output is minimized separately in order to produce an RV-testable circuit, the circuit requires nine product terms. This RV-testable circuit is shown in Figure 10(c). There are 11 gates shown, but note that some of the terms can be shared. Gate q_3 is shared between output g and h, and gate q_4 is shared between outputs g and f.

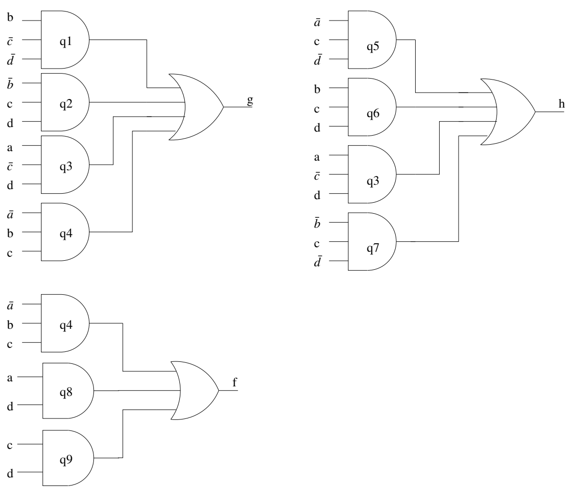
Example 2 shows a way to implement a two-level delay-verifiable circuit. Reference [7] also provides a program



(a) Multi-output function minimized with ESPRESSO



(b) Delay-verifiable circuit



(c) RV-testable circuit

Figure 10. Example 2: Implementing a delay-verifiable circuit

that can implement a multi-level delay-verifiable circuit. Another implementation procedure is explained in [8]. In order to achieve a delay-verifiable circuit extra inputs are created and added to the circuit to provide testability. This method increases the testability and also reduces the hardware required by an RV-testable method [8]. The obvious drawback of this method is that it requires extra inputs to the circuit.

4.4. Conclusion

In order to provide better test coverage for path delay faults in combinational circuits, the concept of delay-verifiable circuits has been presented. Example 1 shows that a non-RV-testable circuit with non-prime literals can still be verified to function correctly (temporally). Furthermore, Example 2 has demonstrated a way to implement a delay-verifiable circuit.

5. Variable sampling times

Delay fault testing is necessary in order to verify that a circuit is functional at its specified operating speed. Therefore the operating frequency that can be guaranteed is strongly dependent on the amount and quality of the delay fault testing.

The following is based on reference [4] section V:

The robust delay of a circuit, τ_R, is calculated by the following linear program:

$$\max\left(\left(\sum_{e\in Q}\delta(e),\, \forall Q \notin (\mathcal{P}\cup\mathcal{R})\right), d\right)$$

subject to:

$$\sum_{e\in P}\delta(e) \leq d, \quad \forall P \in \mathcal{P}$$

where $\mathcal{P}$ is the set of RPDFs in the circuit and $\mathcal{R}$ is a maximum RD set. d is the sampling period.

That is, we can guarantee that the circuit – even along the untestable paths – does not contain any delay faults larger than τ_R if we can verify that the RPDFs stay within these bounds.

Reference [4] gives an example for the above circuit. We adapt this example for our purposes using the approximate delays for a 0.25μm standard cell process. The RPDF paths for both rising and falling transitions denoted by $\mathcal{P}$ are $\{a,e,f,h,k,n\}$, $\{a,e,f,g,j,m\}$, $\{b,d,f,h,k,n\}$, $\{b,d,f,g,j,m\}$, $\{c,g,k,n\}$, $\{c,g,j,m\}$, $\{b,d,i,l\}$, $\{a,d,i,l\}$. All gates are assumed to have a nominal delay in the interval $[d^P_{min}, d^P_{max}] = [0.08\,\text{ns}, 0.11\,\text{ns}]$, the sampling time is accordingly set to $d = \max(d^P_{max})\forall P \in \mathcal{P} = 0.44\,\text{ns}$. The following linear program results:

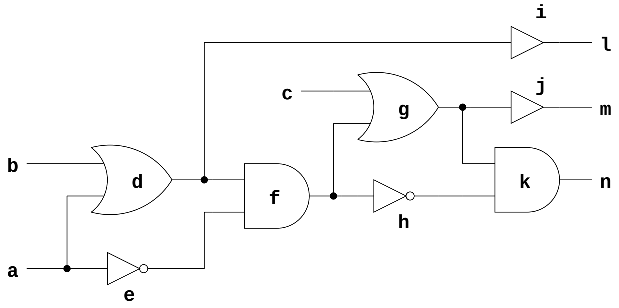


Figure 11. Random observation time example circuit

$$\tau_R = \max\left(\sum_{x \in Q} \delta(x), 0.44\,\text{ns}\right), \quad \forall P \notin \mathcal{P}$$

subject to:

$$\begin{aligned}
\delta(a,e)+\delta(e,f)+\delta(f,h)+\delta(h,k)+\delta(k,n) &\leq 0.44\,\text{ns}\\
\delta(a,e)+\delta(e,f)+\delta(f,g)+\delta(g,j)+\delta(j,m) &\leq 0.44\,\text{ns}\\
\delta(b,d)+\delta(d,f)+\delta(f,h)+\delta(h,k)+\delta(k,n) &\leq 0.44\,\text{ns}\\
\delta(b,d)+\delta(d,f)+\delta(f,g)+\delta(g,j)+\delta(j,m) &\leq 0.44\,\text{ns}\\
\delta(c,g)+\delta(g,k)+\delta(k,n) &\leq 0.44\,\text{ns}\\
\delta(c,g)+\delta(g,j)+\delta(j,m) &\leq 0.44\,\text{ns}\\
\delta(b,d)+\delta(d,i)+\delta(i,l) &\leq 0.44\,\text{ns}\\
\delta(a,d)+\delta(d,i)+\delta(i,l) &\leq 0.44\,\text{ns}
\end{aligned}$$

Yielding $\tau_R = 1.32\,\text{ns}$, that is, if the RPDF paths are verified to have a delay $\leq 0.44\,\text{ns}$, the circuit is verified to have a delay no longer than $1.32\,\text{ns}$.

Lam et al. [4] prove, that by additionally verifying a lower bound, better guarantees on the circuit performance can be given. (Note that verifying the lower bound is done in order to reduce the solution space of the linear program and thereby catch slow devices, not in order to catch paths that are too fast.) The following linear program calculates an upper limit on the circuit delay:

$$\tau_R \leq \max\left(\left(\sum_{e \in Q} \delta(e), \forall Q \notin (\mathcal{P} \cup \mathcal{R})\right), d\right)$$

where

$$d = \max(d^P_{max}) \forall P \in \mathcal{P}$$

subject to:

$$d^P_{min} \leq \sum_{e \in Q} \delta(e) \leq d^P_{max}, \quad \forall P \in \mathcal{P}$$

In the example this gives us:

$$\max\left(\sum_{x \in Q} \delta(x), 0.44\,\text{ns}\right), \quad \forall P \notin \mathcal{P}$$

such that

$$\begin{array}{lcl}
0.32 \leq & \delta(a,e)+\delta(e,f)+\delta(f,h)+\delta(h,k)+\delta(k,n) & \leq 0.44\\
0.32 \leq & \delta(a,e)+\delta(e,f)+\delta(f,g)+\delta(g,j)+\delta(j,m) & \leq 0.44\\
0.32 \leq & \delta(b,d)+\delta(d,f)+\delta(f,h)+\delta(h,k)+\delta(k,n) & \leq 0.44\\
0.32 \leq & \delta(b,d)+\delta(d,f)+\delta(f,g)+\delta(g,j)+\delta(j,m) & \leq 0.44\\
0.16 \leq & \delta(c,g)+\delta(g,k)+\delta(k,n) & \leq 0.22\\
0.16 \leq & \delta(c,g)+\delta(g,j)+\delta(j,m) & \leq 0.22\\
0.16 \leq & \delta(b,d)+\delta(d,i)+\delta(i,l) & \leq 0.22\\
0.16 \leq & \delta(a,d)+\delta(d,i)+\delta(i,l) & \leq 0.22
\end{array}$$

(All values in ns.) Resulting in $\tau_R = 0.56\,\text{ns}$.

5.1. Tester limitations

Ideally a circuit under test would be verified exactly against these bounds, that is, for the example it should be verified that the first four of the paths above are no shorter than $0.32\,\text{ns}$ and no longer than $0.44\,\text{ns}$, whereas the four remaining paths stay within the interval $[0.16\,\text{ns}, 0.22\,\text{ns}]$. If this is done, the circuit can be guaranteed to have a delay of at most $0.56\,\text{ns}$.

Unfortunately, chip testers not only provide limited accuracy in the positioning of the sampling time, they also offer no more than a few different sampling times for each pin. The above method remains, but since the bounds that can be verified probably will be less tight, only a lower speed can be guaranteed.

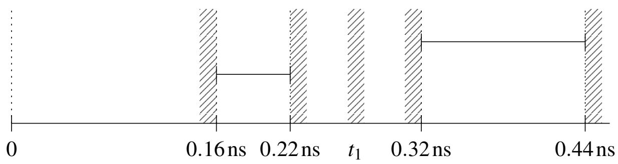


Figure 12. Verification of upper and lower bounds.

Figure 12 shows the problems associated with the limited sampling precision of chip testers. For example, a certain state of the art chip tester [1], offers 45 ps accuracy in the positioning of the timing edge. Using these worst case assumptions, the speed guarantees given for the example circuit drop to 1.01 ns.

Furthermore, the tester allows for only 6 timing edges per pin. Assuming, that only three instead of four timing edges can be allocated to the test of the example circuit, sampling could occur at time t_1 instead of at both 0.22 ns and 0.32 ns. Choosing t_1 at the middle point between the two lets the timing guarantee for the circuit further drop to 1.02 ns, 82% slower than the ideal case. Of course, the effect of this would have been worse for a case where the two sampling times that were substituted by a single sampling time were further apart.

5.2. Consequences

It has been shown that verifying path delays is a powerful way to achieve guarantees for circuit performance. Unfortunately, this method is very sensitive to the accuracy and number of sampling times supported by the tester hardware. This creates a trade-off between the cost of the tester hardware and the speed of the tested circuit.

One way to reduce the cost associated with delay fault testing could be to make assumptions about the lower bound on circuit delay. This lower bound was introduced in order to reduce the solution space of the linear program, as it otherwise tends to find solutions where many of the delays are zero. This is hardly likely to occur. In some circuits it is possible that a fast path can cause another path to be slowed down. If one dared assume that this is a sufficiently rare type of delay fault and accept the risk of allowing a device with this kind of delay fault pass, the test for the lower bound could be replaced by a worst-case assumption.

6. Conclusion

This paper has explained several aspects of delay fault testing. Two models exist for delay faults: the gate delay model and path delay model. The gate delay model has the advantage that it is linear with the number of gates in a circuit. The path delay fault model more accurately models the delays in a circuit, and can better verify the circuit's timing. However, the path delay model has worst-case exponential growth with the number of gates. In addition, this paper has presented the principles of test pattern generation using the path delay model. A five-valued logic system can be used to model the faults in a circuit. An algorithm has been presented to minimize the resultant test set, as well as ideas for parallel testing of delay faults. In addition, the concept of delay-verifiable circuits has shown that not all paths in a circuit need to be tested in order to verify its correctness. A procedure to implement delay-verifiable circuits was explained. Finally, this paper examines the impact of the tester hardware on delay fault testing. The limited accuracy of the testing hardware is shown to weaken the guarantees given on circuit performance.

Delay fault testing is an increasingly important aspect of verifying a circuit's functionality. There are several trade-offs to consider when devising the test procedure for a circuit. First, the cost of time and money for delay tests are high. Delay faults may be detected with other testing methods (i.e stuck-at fault testing), but temporal correctness cannot be guaranteed. Also, the cost of tester hardware must be weighted against the desired speed of the circuit. These trade-offs must be considered in order to provide the required test coverage of a circuit at the lowest cost.

References

[1] Advantest American Corporation. T6683 – SOC test system. `http://www.advantest.com/aac/t6683.pdf`, june 2001.

[2] V. S. Iyengar, B. K. Rosen, and I. Spillinger. Delay test generation – algebra and algorithms. *IEEE International Test Conference*, pages 867–876, 1988.

[3] V. S. Iyengar, B. K. Rosen, and I. Spillinger. Delay test generation – concepts and coverage metrics. *IEEE International Test Conference*, pages 857–866, 1988.

[4] W. K. Lam, A. Saldanha, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Delay fault coverage, test set size, and performance trade-offs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD*, 14(1):32–44, january 1995.

[5] C. J. Lin and S. M. Reddy. On delay fault testing in logic circuits. *IEEE TCAD*, 6/5:694–703, september 1987.

[6] P. Menon and W. Ke. Synthesis of delay-verifiable two-level circuits. *European Design and Test Conference*, 1(28):297 – 301, Feb. 1994.

[7] P. Menon and W. Ke. Synthesis of delay-verifiable combinational circuits. *Computers, IEEE Transactions on*, 44(1):213 – 222, Feb. 1995.

[8] X. Yu and Y. Min. Design of delay-verifiable combinational logic by adding extra inputs. *Test Symposium, 1997. (ATS '97) Proceedings., Sixth Asian*, 1(17-19):332 – 336, Nov. 1997.