Hazards in Combinational Networks

So far we have been concerned with the steady state behavior of combinational networks, i.e., after all inputs, internal connections and outputs have achieved steady state.

In practice, there are delays that can affect the transient of the network between steady-states. The effects of such delays are called hazards.

We next consider the following questions:

- What kind of hazards exist?
- What causes them?
- How can we analyze them?
- How can we avoid them?
Two sources of trouble:

- the input signals varying in the amount of time they take to achieve steady-state at the next input bit sequence of interest

- varying delays along different paths through the network due to gates settling to steady-state at different rates and different propagation times between gates

Two types of trouble:

- static hazard: the steady-state output of the network is the same for two input patterns, but during transition the output changes states and then returns to the desired steady-state value (in general it changes state an even number of times before settling)

- dynamic hazard: the steady-state output of the network is different for two input patterns, but during transition the output changes state an odd number of times before settling
static 0–hazard

static 1–hazard

dynamic hazards
Function Hazards

Suppose our network settles instantly given any change in the input.

Also suppose the two input sequences for which we want the steady state values differ in multiple bits.

If the bits change at different times then the network output may take on transient incorrect values even though it settles immediately after a single bit changing.

Such faults are independent of the network used to implement the switching function.
<table>
<thead>
<tr>
<th>WX</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
<td>4</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>5</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>6</td>
<td>14</td>
<td>10</td>
</tr>
</tbody>
</table>

0000 to 1100

steady-state is 1 in both cases

w 0 to 1 then x 0 to 1 0000 1000 1100
output 1 1 1
no hazard

x 0 to 1 then w 0 to 1 0000 0100 1100
output 1 0 1
function static 1–hazard

function dynamic hazard

x 1 to 0, then z 0 to 1, then y 1 to 0

0110 0010 0011 0001
0 1 0 1
4 input AND gate

function static 0–hazard
for any multiple bit change
that passes through (1, 1, 1, 1)

9 to 13 to 15 to 7

no function static 1–hazards

no function dynamic hazards

(it is assumed that the multiple
bit changes are distinct bits
each changing once, not
one bit changing multiple
times)

<table>
<thead>
<tr>
<th>WX</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>5</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>6</td>
<td>14</td>
<td>10</td>
</tr>
</tbody>
</table>

4 input OR gate

function static 1–hazard
for any multiple bit change
that passes through (0, 0, 0, 0)

no function static 0–hazards

no function dynamic hazards

(it is assumed that the multiple
bit changes are distinct bits
each changing once, not
one bit changing multiple
times)
The effect of function hazards can be eliminated in various ways (when they are important).

For example, combinational logic is used to set the input lines to the memory devices in sequential machines. Therefore, transient false outputs must not be allowed to cause a transition into an incorrect state.

- We could ignore the transient outputs until they settle and then allow the resulting signals to propagate downstream in the machine. This is done in Pulse mode sequential machines (synchronous). State transitions are clocked. So the effect of the input signals on the memory devices is only seen when they are known to be stable, i.e., the clock period is longer than worst case hazard settling time.

- We could also allow only one input bit to change at a time and let the output settle for each. This removes the uncertainty in the transient behavior. The machine must then be designed to accept the transient outputs as intermediate states and, transition through a series of them into the desired final state. This is the way Fundamental mode (asynchronous) sequential machines are designed.

Both of these will be discussed later. We will assume that the input to the combinational circuit changes in only one bit and analyze the resulting hazards.
Logic Hazards

We will assume that our networks are controlled so that only one bit can change in the input pattern.

Suppose we have double-rail logic. It is reasonable to assume for some technologies that for a single input bit change, \( x \) and \( x' \) are available essentially simultaneously. This is not always the case however. For example, \( x \) and \( x' \) could be output lines of a flip-flop which do not change at exactly the same time. Or \( x' \) could be the output of an inverter gate that has, for some reason, a long delay. We will model this situation by a network that responds instantly and \( x' \) being computed by an inverter that has appreciable delay.

This can be viewed as a transitional form between function and logic hazards. It also gives insight into the cause of static hazards due to single bit changes.
\[ f(w, x, y, z) = x'y'z + wyz + wxz + w'xy \]

(1, 1, 1, 1) to (0, 1, 1, 1) with a transient state

where \( w = 0 \) and \( w' = 0 \) simultaneously

\[ f(1, 1, 1, 1) = 1*1*0 + 1*1*1 + 1*1*1 + 0*1*1 \]

\[ \text{transient} = 1*1*0 + 0*1*1 + 0*1*1 + 0*1*1 = 0 \]

\[ f(0, 1, 1, 1) = 1*1*0 + 0*1*1 + 0*1*1 + 0*1*1 \]

static 1-hazard due to delay which causes \( w' = w \)
Now suppose we are in a technology where it is reasonable to expect $x$ and $x'$ to be available simultaneously for a single input bit change, or that we have a single-rail network.

Hazards can still occur but we have removed all difficulty with the inputs by the assumptions above so we must look at the effect of delays in the network itself. Such hazards are called logic hazards.

The key here is to have a situation where the delays in the network make it look like one of the inputs is temporarily equal to its complement, i.e., the complement is implicitly computed and delayed (or advanced) relative to the effect of the uncomplemented variable.
Example of a Logic Hazard

Suppose the delay of gate A is less than the delay of gate B.

Pattern 1 \((x, y, z) = (0, 0, 0)\)
Pattern 2 \((x, y, z) = (0, 0, 1)\)
Note that the AND gate with output labelled C has seen a multiple input bit change with network delays causing a time skew in the changes.

\[
\begin{align*}
\text{start} & \quad A = 0 \quad B = 1 \\
\text{transient} & \quad A = 1 \quad B = 1 \\
\text{end} & \quad A = 1 \quad B = 0
\end{align*}
\]

Note that in the two steady states \( A = B' \) and in the transient that causes the difficulty \( A = A' \), i.e., two signals that should be complements are temporarily the same.

This is essentially the equivalent of a function hazard of the AND gate which is the output of the network caused by a single input bit change and delays in the flow of information through the network.

This can also be related back to one of the inputs and its complement. Obviously, this input is \( z \) since it is the one transitioning.

For the particular input patterns and transient we have

\[
\begin{align*}
A & = (y + z) \\
& = (0 + z) \\
B & = (x + z') \\
& = (0 + z') \\
C & = (z \cdot z')
\end{align*}
\]
Note that the gate that experiences the multiple input bit changes does not have to be the final gate. It could be internal. But in that case, the assignments to the rest of the variables must create a path to the output gate that allows the static hazard to show up in transient values of \( f \).

Also note that if the delay of gate \( A \) was much larger than \( B \), \( z' \) would have changed to 0 before \( z \) changed to 1 and the output would not have experienced a hazard, i.e., \((z, z') = (0, 1) \rightarrow (0, 0) \rightarrow (1, 0)\).

Our analysis techniques will find conditions under which **there exists at least one set of delays** that could be assigned to gates on particular paths through the network that will cause a hazard.

The design techniques that result will guarantee that **there is no set of delays** that could be assigned to the gates in the network to cause hazards.

As a result, the techniques are conservative with respect to a particular set of delays in a particular network.
The identification of \( x = x' \) as the source of static 0-hazards can be made more general by considering a multiple input AND gate (and a multiple input OR gate for static 1-hazards) under the assumption that no upstream static or dynamic hazards are present in the network.

The previous conclusions about function hazards for these multiple input gates let us conclude that the only way a static hazard can occur on the output of such a gate is to have \( x = x' \) transiently.

The two results can be combined to show the same result for an arbitrary gate in the network, i.e., one whose function is other than AND or OR.

It can also be seen from these results that delays of the propagation of the change to \( x \) does not cause static hazards (although we will revisit such delays when discussing dynamic hazards).
The examples given so far have created static logic hazards by making delays cause the evaluation of a function (that is not \( f \), but is related to \( f \) and the network) that effectively views the input bit \( x \) that has changed and \( x' \) as two distinct switching variables which temporarily have the same value. Once steady state is reached the normal complement relationship is restored, i.e., \( x' = (x)' \).

We need a way to analyze this situation. Specifically, we need a way to construct the function that depends upon \( f \) and the particular network we have and tells us when the network can emit transient incorrect values.

Suppose we define a switching function \( F \) that is a function of \( 2n \) variables \( v = (x_1, x'_1, \ldots, x_n, x'_n) \), i.e., view \( x_i \) and \( x'_i \) as completely independent variables.

Values of \( v \) where \( x'_i \neq x_i \) represent network steady states and we want

\[
F(v) = f(x_1, x_2, \ldots, x_n)
\]

For values of \( v \) where there is exactly one \( x_i \) such that \( x'_i = x_i \) we want \( F(v) = 1 \) if there exists a set of delays that causes the network to output a 1 during the transition.
• How do we find $\mathcal{F}$?

• How does it allow us to detect static hazards?

We can find $\mathcal{F}$ by using all of our earlier network analysis theorems, with one modification.

We lose the basic postulates concerning $x$ and $x'$. We can no longer use the following facts:

\[
xx' = 0 \\
x + x' = 1
\]

Also ANY THEOREM DEDUCED FROM THESE POSTULATES CANNOT BE USED.

When analyzing the networks we can only use:

• Distribution

• Idempotence

• Absorption

• DeMorgan can be used with care. (You have to decide what complements do not cost time and what ones need to be modeled. A conservative approach can cause an explosion in the number of signals that you are viewing as distinct variables.)
Consider our previous example.

We had

\[ F(x, x', y, y', z, z') = (x + z')(y + z) \]
\[ = xy + xz + yz' + zz' \]

We cannot remove the last term since we treat \( z \) and \( z' \) as separate variables.

The logic static 0-hazard was seen with the following three patterns for \((x, x', y, y', z, z')\): \((0, 1, 0, 1, 0, 1)\), \((0, 1, 0, 1, 1, 1)\) and \((0, 1, 0, 1, 1, 0)\). Evaluating \( F \) at these points yields:

\[ F(0, 1, 0, 1, 0, 1) = 0 + 0 + 0 + 0 \]
\[ F(0, 1, 0, 1, 1, 1) = 0 + 0 + 0 + 1 \]
\[ F(0, 1, 0, 1, 1, 0) = 0 + 0 + 0 + 0 \]

as desired.
The terms in which a variable and its complement do not appear are set by selecting $x$ and $y$ (in general all variables not changing) so that they evaluate to 0 independent of the value of $z$ (in general the input that is changing). Note that they identify the paths from $z$ to the output on which the delays must be set to get a 0-hazard.

The term that involves $z$ and $z'$ determines what conditions cause a transient 1. For this example the term is trivial. in general there may be more than one and they may involve variables that are not transitioning. (This determines whether or not there is a path from an internal gate emitting the static hazard to the output of the network.)
Example 2

\[ w'x(wy' + z) + wy \]
This yields

\[ F = wy + w'x(\text{\textit{wy}}' + z) \]
\[ = \text{\textit{wy}} + w'xz + w'w'xy' \]

\((w, x, y, z) = (d, 1, 0, 0)\) sets the first two terms to 0 for two steady state assignments and sets the third to \(w'w\).

It also identifies the two of the three possible paths through which \(w\) can affect the output. (The path through \(C\) does not affect the output since \(y = 0\).)

The transition \((1, 1, 0, 0) \rightarrow (0, 1, 0, 0)\) with a transient condition of \(w = w' = 1\) causes a logic static 0-hazard. Essentially, \(w'\) rises to 1 before the effect of \(w\) dropping to 0 is felt via the path to D through A and B.

(It is also possible to have the hazard occur for the transition \((0, 1, 0, 0) \rightarrow (1, 1, 0, 0)\) but it is less likely physically.)
So far the $\mathcal{F}$ functions we have seen have not required any reduction. In some cases, direct analysis of the network yields a SOP which can be reduced via absorption and identity while viewing $x_i$ and $x'_i$ as distinct switching variables. This results in a final SOP that can be reduced no further and has terms similar to those seen in our examples, i.e., some with only a variable or its complement appearing and some with both.

These terms are fundamental to the analysis of static hazards.

Consider the following example:

$$
\mathcal{F} = x'xy + xyy' + xyw + y'zx' + y'z + y'zw \\
= x'xy + xyy' + xyw + y'z
$$

Two terms were eliminated via absorption with $y'z$.

Note that if we view $\mathcal{F}$ as a function of $2n$ variables all variables in the initial SOP are monoform. Therefore reduction via absorption and identity is equivalent to Tison’s first method and the terms that remain are the prime implicants of $\mathcal{F}$. 
Note that those terms which involve a variable and its complement are not important for steady state analysis and can be removed when relating the PI’s of $F$ to the implicants of $f$. However, when they are removed the remaining terms are not necessarily the prime implicants of $f$.

Consider $F = wxy + w'xy$ from a simple two-level AND/OR network. We have $f = xy$ which is a single PI.

The PI’s of $F$ can be defined in an equivalent fashion and used to formalize the search for static hazards.
**Definition:** A 1-set of a network is a set of literals such that:

(a) Whenever all of the literals are set to 1, the network output is 1. (Recall that we treat $x$ and $x'$ as separate switching variables so each can be set independent of the other.)

(b) If any literal is removed from the set, (a) no longer holds.

For our example, since we were able to reduce to a simple SOP,

$$\{w, y\} \quad \{w', w, x, y'\} \quad \{w', x, z\}$$

are 1-sets.

**Definition:** If $x$ and $x'$ appear in the same 1-set, $x$ is called an unstable variable and the 1-set is called an unstable 1-set, e.g., $\{ww'xy'y\}$.

**Definition:** A stable 1-set does not contain any unstable variables, e.g., $\{w, y\}$ and $\{w', x, z\}$. 
**Definition:** A 1-set is active for an input pattern if all of the literals in the 1-set are set to 1 in the input pattern, e.g., \( \{w', x, z\} \) is active for \( w = 0, x = z = 1 \).

Note that since an input pattern only sets uncomplemented variables and complemented variables are evaluated accordingly, an unstable 1-set can never be active for a single input pattern, i.e., either \( w \) or \( w' \) must be 0.

**Definition:** An unstable 1-set with a single unstable variable is active for a transition from one input pattern to another input pattern that differs in the unstable variable if all literals involving stable variables are set to 1 by the input patterns, e.g., \( \{ww'xy'\} \) is active for a transition from any pattern with \( x = 1 \) and \( y = 0 \) and \( w \) changing from 0 to 1 or from 1 to 0.

Note that an unstable 1-set with two or more unstable variables cannot be active for a single bit transition. The transition would involve one of the unstable variables and the rest would generate a literal that evaluates to 0. For example, \( \{w, w', x, x', y\} \) cannot be active for any transition by \( w \) since \( x \) cannot also change due to single bit restriction.
The reasoning we used earlier to create logic static 0-hazards can now be stated formally as the following theorem.

**Theorem:** A static logic 0-hazard exists in a network if and only if there exists a pair of inputs such that for both inputs the network outputs a 0 and an unstable 1-set which is active for the transition.

In our example, the unstable 1-set is active for the transition

\[(1, 1, 0, 0) \rightarrow (0, 1, 0, 0)\]
Suppose we have a two-level AND/OR network using double-rail logic. Static 0-hazards are not a problem.

**Theorem:** No logic static 0-hazards occur in a two-level AND/OR network.

**Proof:** The products that are input to the AND gates at the first level are all assumed to be i-cubes, i.e., there is no gate which has as input both a variable and its complement. (This is reasonable since such a gate has no steady state value to the network.) Therefore the expression for $f$ is the same as the intitial expression for $F$. THEY ARE EVALUATED DIFFERENTLY HOWEVER. Remember that $f$ is evaluated only at steady state assignments where $x_i \neq x'_i$ for all switching variables. When $F$ is evaluated $x_i = x'_i$ may hold. This is important when detecting static 1-hazards.

It may be possible to reduce the SOP expression for $f$ and $F$ via absorption since we have not assumed that the network is a minimal realization. However, since no product in the expression contained a variable and its complement and none of the reductions will create one, there can be no unstable 1-sets.

Therefore, there are no static 0-hazards. □.
It is also possible to deduce this directly from the behavior an AND gate which must evaluate to 0 for the two steady state assignments that differ by one bit. Effectively, all AND gates must be 0 independent of the bit that is changing so the OR gate does not see any changes in its input bits and the output of the network stays 0 during the transition of any input to the network, $x_i$. 