Hazard is a momentary \textit{unwanted} switching transient at a logic function’s output (i.e., a glitch).

Hazards/glitches occur due to unequal propagation delays along different paths in a combinational circuit.

Can take steps to try and eliminate hazards.

There are two types of hazards; \texttt{static and dynamic}.

For asynchronous circuits in particular, hazards can cause problems in addition to other issues like races and non-fundamental mode operation!

\textbf{Momentary false logic function values in an asynchronous circuit can cause a transition to an incorrect stable state!}
Static hazards

- Static-0 Hazard:
  - Occurs when output is 0 and should remain at 0, but temporarily switches to a 1 due to a change in an input.

- Static-1 Hazard:
  - Occurs when output is 1 and should remain at 1, but temporarily switches to a 0 due to a change in an input.

![Static-0 hazard (0->0)](image)

![Static-1 hazard (1->1)](image)
Dynamic hazards

Dynamic Hazard:

- Occurs when an input changes, and a circuit output should change 0 -> 1 or 1 -> 0, but temporarily flips between values.

![Dynamic hazard diagrams](image-url)

- Dynamic hazard (0->1)
- Dynamic hazard (1->0)
Illustration

- Consider the following circuit with delays where only one input (input b) changes...

- Draw a timing diagram to see what happens at output with delays.

- From the logic expression, we see that b changing should result in the output remaining at logic level 1...

- Due to delay, the output goes 1->0->1 and this is an output glitch; we see a static-1 hazard.
Fixing hazards (2-level circuits) (1)

- When circuits are implemented as **2-level SOP (2-level POS)**, we can detect and remove hazards by inspecting the K-Map and *adding redundant product (sum) terms*.

  \[
  f = ab + b'c
  \]

- Observe that when input \( b \) changes from 1->0 (as in the previous timing diagram), that we “jump” from one product term to another product term.

  - *If adjacent minterms are not covered by the same product term, then a HAZARD EXISTS!!!*
Fixing hazards (2-level circuits) (1)

\[ f = ab + b'c + ac \]

- The extra product term does not include the changing input variable, and therefore serves to prevent possible momentary output glitches due to this variable.
Fixing hazards (2-level circuits) (3)

- The redundant product term is not influenced by the changing input.

\[
f = ab + b'c + ac
\]
Fixing hazards (2-level circuits) (4)

- For 2-level circuits, if we remove all static-1 hazards using the K-Map (adding redundant product terms), we are guaranteed that there will be no static-0 hazards or dynamic hazards.

- If we work with Product-Of-Sums, we might find static-0 hazards when moving from one sum term to another sum term. We can remove these hazards by adding redundant sum-terms.
Hazards in asynchronous circuits

Consider our first circuit with a hazard, but assume it is not combinatorial, but rather asynchronous.

We can draw the transition table, and see that there is the potential to end up in an incorrect stable state.

\[
y = ab + b'y
\]

<table>
<thead>
<tr>
<th>curr state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab=00</td>
<td>0 1 1 1 0</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>Y Y Y Y Y</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0 0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1 0 1 1</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram:
- A circuit diagram showing the logic gates and the transition table.
Hazards in multi-level circuits (1)

- 2-level circuits are easy to deal with and hazards can be removed...

- The situation is harder with multi-level circuits in which there are multiple paths from an input to an output:
Hazards in multi-level circuits (2)

- Timing diagram shows output changing 0->1->0->1.

- Hazards like this are hard to fix. We could always find a 2-level circuit of the previous circuit and get something hazard free...
Fixing hazards with latches (1)

- Can also fix hazards using SR or S’ R’ latches.
  - An SR Latch can tolerate momentary 0s appearing at its inputs (since we might momentarily move from a set or reset to a hold and then back).
  - An S’ R’ Latch can tolerate momentary 1s appearing at its inputs (since we might momentarily move from a set or reset to a hold and then back):

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>̅Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>̅Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Consider our original circuit with a static-1 hazard (temporary 0 at output):

\[ f = ab + b'c \]

<table>
<thead>
<tr>
<th>( bc )</th>
<th>( 00 )</th>
<th>( 01 )</th>
<th>( 11 )</th>
<th>( 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td>( 1 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( 1 )</td>
<td>( 0 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
</tr>
</tbody>
</table>
Fixing hazards with latches (3)

- Consider that we take our output \( f \) from the output of a latch.

- Since we are trying to fix static-1 hazards we need to be able to tolerate momentary 0s at latch inputs \( \Rightarrow \text{Use a SR Latch (NOR Latch)}. \)

- To get the function \( f \) from the latch output, we need equations for \( S \) and \( R \) of the latch (so that the latch gets SET when \( f \) should be one, otherwise RESET).

\[
\begin{array}{c|cccc}
\text{bc} & 00 & 01 & 11 & 10 \\
\hline
\text{a} & 0 & 0 & 1 & 0 & 0 \\
& 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]  
\text{Equation for S}

\[
S = ab + b'c
\]

\[
\begin{array}{c|cccc}
\text{bc} & 00 & 01 & 11 & 10 \\
\hline
\text{a} & 0 & 1 & 0 & 1 & 1 \\
& 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]  
\text{Equation for R}

\[
R = b'c' + a'b
\]
Fixing hazards with latches (5)

- Draw a circuit using the latch, and see that glitch in output due to the hazard is gone.
Output assignment in asynchronous circuits

- Flow and transition tables might have unspecified entries for circuit outputs.
  - This might be a result of the fundamental mode assumption.
  - This might be a result of unstable states.

- Note: we always have output values assigned for stable states!

- We should think about what happens with the unspecified output values...
  - They are, in effect, don’t cares that we can exploit during minimization of the output logic equations.
  - But, we might temporarily pass through these values while transitioning from one stable state to another stable state.

- Depending on the output equations that we derive (due to minimization of the output equations), we might end up having glitches at our circuit outputs.
  - Glitches are bad; they could get fed into another circuit causing problems. They also waste power.
Avoiding output glitches

Consider the following flow table with don’t cares at some outputs (circuit has one input and one output):

<table>
<thead>
<tr>
<th>curr state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x=0</td>
<td>x=1</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>-</td>
</tr>
</tbody>
</table>

Consider a transition between two stable states due to a change in an input value and how it might be best to assign the don’t care value in an unstable intermediate state:

- If both stable states produce a 0 output, make output 0 instead of a don’t care.
- If both stable states produce a 1 output, make output 1 instead of a don’t care.
- If stable states produce different outputs, the output can remain a don’t care and be used to find a smaller output circuit.

This will enable us to avoid output glitches when passing through unstable temporary states.
Example

- If we consider possible transitions, we see that some of the output don’t cares should be changed to 0 or 1 to avoid glitches.

<table>
<thead>
<tr>
<th>curr state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x=0</td>
<td>x=1</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>1</td>
</tr>
</tbody>
</table>

- Next state and output changes:

<table>
<thead>
<tr>
<th>curr state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x=0</td>
<td>x=1</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>-</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>1</td>
</tr>
</tbody>
</table>

The above changes will avoid temporary glitches at the outputs during transitions where the output should not change.