Asynchronous Circuits

Races, Cycles and Effect of Hazards

"All circuits have problems, but asynchronous circuits have more problems"

The Difference Between Asynchronous and Synchronous Feedback

- Synchronous feedback must wait for the clock.
- Input signals must not change when the clock does.
- Always behave like the state table says

- Asynchronous feedback comes immediately with only gate delays.
- Inputs can come at any time.
- Circuits may not behave as the state table says.

FIG. 2-1 Illustrates synchronous and asynchronous feedback

Asynchronous feedback: the only delay in the feedback loop is the propagation delay of the gates.

Synchronous feedback: The feedback is delayed until the active clock edge.
The 3 Forms of Asynchronous Feedback

**FIG. 2-2** The three (one is fake) forms of asynchronous feedback circuits

- **OSCILLATOR**
  - An odd number of inversions in loop.
  - Output inverts input.
  - Circuit oscillates.

- **LATCH**
  - An even number of inversions.
  - The output reinforces the input.
  - If $Q_L$ was originally high, it stays high.
  - If $Q_L$ was originally low, it stays low.
  - The circuit remembers
  - It is a latch.

- **NO FEEDBACK**
  - This circuit remembers.
  - But only when $S=0$ and $R=1$

Asynchronous feedback circuits **oscillate or remember**.
They **oscillate** if the feedback loop has an odd number of inversions.
They **remember** if the feedback loop has an even number of inversions.

---

Sensitized Paths

A signal path is **sensitized** if a change in its input changes its output.

**FIG. 2-3** Sensitized and unsensitized paths

- **SENSITIZED PATH**
  - A signal change goes through both AND and OR gates.

- **UNSENSITIZED PATH**
  - A signal change cannot go through the AND gate.

A feedback path must be sensitized for the circuit to remember/oscillate.
In practical circuits, the feedback is only sensitized part of the time.

**FIG. 2-4** Apparent feedback path is never sensitized

- **SENSITIZED PATH**
  - This circuit remembers.
  - When $S=0$ and $R=1$
  - A signal change can propagate around the loop

- **UNSSENSITIZED PATH**
  - $X=0$ shuts off the AND gate.
  - $X=1$ shuts off the OR gate.
  - There is never a sensitized feedback loop!
Analysis of Asynchronous Circuits

Review of Synchronous State Machines

The state is the collective output of all flip-flops and latches. The next state is the state after the clock edge.

**FIG. 2-5** A state machine using D flip-flops.

The present state is value of “ABC” the present Q outputs

The next state is the value of “a+b+c” the present D inputs.

A state machine is defined by its *next state table.*

**FIG. 2-6** State Graph and Next State Table for a Mythological Circuit.

<table>
<thead>
<tr>
<th>State</th>
<th>Next State</th>
<th>a+b+c</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>x=0</td>
<td>001</td>
</tr>
<tr>
<td></td>
<td>x=1</td>
<td>011</td>
</tr>
<tr>
<td></td>
<td>x=0</td>
<td>010</td>
</tr>
<tr>
<td></td>
<td>x=1</td>
<td>110</td>
</tr>
</tbody>
</table>

Only 4 of the 8 states are shown here.

Asynchronous Circuits

State Variables For Asynchronous Circuits

In synchronous circuits (with D flip-flops)
- Each flip-flop can remember one bit.
- Each flip-flop holds a state variable.
- The present state is the flip-flop output(s) “A”
- The next state is the D input(s), “a+”

In asynchronous circuits
- Each feedback loop can latch one bit
- Each feedback loop holds a state variable.
- The state is the value fed back to the input.
- The next state is value out of the gate.

Asynchronous circuits the state is value on the feedback lead.
Mark an X on the wire where you want to read the state.
Write “A”, the present state on the output side.
Write “a+”, the next state, on the input side.

**FIG. 2-7** Synchronous and asynchronous state variables

In steady state, the asynchronous present state = next state,  \( A = a^+ \)
On an input change, \( a^+ \) may change.
This will immediately change \( A \) and give a new state.
Relation Between “A” and “a⁺”

**FIG. 2-8** Delay separates a⁺ and A

Asynchronous feedback model.
- a⁺ comes out of circuit.
- After a short delay it reenters the circuit as A
- A travels through the sensitized path and emerges as a⁺.

The state A is continuously regenerated by a⁺.
- a⁺ is continuously recalculated from A and inputs

An input change may change a⁺.
- After the wire delay this will change A.
- A will stay in this new state until an input changes a⁺ again.

We usually put an X on the schematic to show where a⁺ changes to A.
- Should the X be at X₁ or X₂?

* The X is thinking point where a⁺ changes to A. It can be anywhere in the feedback path.

---

**Asynchronous Circuits:**

Relation Between “A” and “a⁺”

**Where To Put The Loop Breaks**

**FIG. 2-9** Break each independent feedback loop with an X (push button).
- An independent feedback loop can store one bit.

**FIG. 2-10** Sometimes two independent loops are really one
- Then cut them with one X.

**FIG. 2-11** Extra breaks:
- Analysis will still work with several breaks in one loop.
- Adds unnecessary states, make more work.
- When in doubt, use extra breaks.

---
**Analysis of an Asynchronous Circuit**

**Tracing the Signals**

A typical analysis:
1. Break the feedback loops at the appropriate places, A and B in FIG. 2-12.
2. Write equations relating the values on the input side of the break ($a^+$ and $b^+$) with values on the output side (A, B) and the inputs (s).
   That is: $a^+ = f(A,B,s)$, $b^+ = g(A,B,s)$.
3. Make a state table.

**FIG. 2-12  A Sample Circuit For Analysis.**

<table>
<thead>
<tr>
<th>State</th>
<th>Next State $a^+$ $b^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>s=0</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

A Sample Circuit For Analysis.

1) Break the circuit at A and B.  
2) Write equations for $a^+$, $b^+$  
3) Make a state table showing the next states for each state

**FIG. 2-13  Tracing State Changes on the Sample Circuit.**

- **a)** Initially $s = 0$; State = AB=00
- **b)** Change s to 1; State = AB=00  
  Soon the next state (01) becomes the state.
- **c)** Travel through the break, state = AB=01
- **d)** Soon the next state $a^+$ will go to 11
- **e)** Now state = AB=11
- **f)** A little later the next state $a^+$ will go to 10
- **g)** Circuit settles down in state = AB=10

The states traveled through are: 00 → 01 → 11 → 10  
00 and 10 are stable states.  
We circle stable states.  
The other states are transient.
Asynchronous Circuits: Tracing the Signals

FIG. 2-14  Tracing State Changes on the Next State Table.

<table>
<thead>
<tr>
<th>State</th>
<th>Next State a⁺ b⁺</th>
<th>Stable states:</th>
<th>Transient intermediate states:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>s=0</td>
<td>where next state = present state.</td>
<td>all other states</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>are circled</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Delays and the Sample Circuit

- The sequence of state changes is independent of the circuit delays.
- With random delays on all gates and all wires, the states would still change 00 -> 01 -> 11 -> 10.
- Be sure s does not change again until the final stable state is reached.
- This circuit is delay independent.

Not all circuits are this well behaved.

Asynchronous Circuits: Circuits with Races

Circuits where delays matter

FIG. 2-15  Circuit where state changes depend on delays

Next-State Equations

\[ a^+ = f(A, B, s) = A B + s \overline{B} \]
\[ b^+ = g(A, B, s) = A B + s \overline{A} \]

Substitute values for A, B and s into the next-state equations, to generate state table data.

<table>
<thead>
<tr>
<th>State</th>
<th>Next State a⁺ b⁺</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B</td>
<td>s=0</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

- This circuit does not always behave like the state table.
- The sequence of states passed through depends on delays.
- We will see:
  - If #1 and #2 are faster, ends in state 10.
  - If #3 and #4 are faster, end in state 01.
  - If they are equal, end in state 11.
- The circuit has a race.
FIG. 2-16 Analysis of a Racy Circuit

a) Initially, $s=0$; State $AB=00$

b) $s$ changes, $s=1$, the next state $a^+b^+=11$
   But only if the delay through gate #1 and #2 equals the delay through #3 and #4.

b) The feedback blocks the output on $sB$ and $sA$ but reestablishes it through $AB$. State change is blocked.

c) $AB=11$ feeds back to three gates.

d) If gates #3 and #4 are slow, after $s$ changes, $s=1$, next state $a^+b^+=10$.

e) $AB=10$ will feed back to give a stable state.

FIG. 2-17 The Analysis of Racy Circuit With Fast Upper Gates.

b) $s$ changes, $s=1$, if gates #1 and #2 are slow, state first changes to $a^+b^+=01$.

c) $AB=01$ will feed back to give a stable state.

FIG. 2-18 Analysis When the Slow Upper Gates

b) $s$ changes, $s=1$, if gates #1 and #2 are slow, state first changes to $a^+b^+=01$. 
Circuits with Races

Analysis of A Racy Circuit Using A State Table

Get next-state equations from the circuit.
Use them to build a state table.
Note any two-bit change in the state.
when in state 00,
s goes 0 → 1
next state is 11
Identify races by:
A two-bit (or more) change in a state variable.
The possible results from the double-bit change is shown in FIG. 2-20.

FIG. 2-19 State-Table and Equations for the Racy Circuit in FIG. 2-17.

Next-State Equations

\[
a^+ = f(A, B, s) = A B + s\overline{B} \\
b^+ = g(A, B, s) = A B + s\overline{A}
\]

Substitute values for A, B and s into the next-state equations, to generate state table data.

TWO BIT CHANGE IN STATE

FIG. 2-20 The Three Possible Outcomes of the Race in the Racy Circuit.

Path in the state-table if delays are equal.
Path when delay for A (via gates #1 and #2) is slower.
Path when delay for B (via gates #3 and #4) is slower.

If gates #1 and #2 are fastest, the stable next state will be AB=10.
If gates #3 and #4 are fastest, the stable next state will be AB=01.
With equal delays, the stable next state will be AB=11.

Races Have a Simultaneous Two-Bit Change in the State Variables

Look for double-bit change in a state variables, and you will find the races.
Noncritical, and Other Non-important Races.

Not all races are a problem:

Noncritical Races

Some races take different paths but end in the same final stable state. These are called noncritical races. FIG. 2-21 and FIG. 2-22

Unused Races ("Who Cares" Races)

Some races are in a part of the state table which is never used. FIG. 2-23

Races To Equivalent States (States that could be merged)

The final states of the race might be equivalent. The machine would externally behave the same no matter how the race came out.

Asynchronous Circuits: © John Knight Electronics Department, Carleton University April 1, 2006

FIG. 2-21 A Noncritical Race With Three Paths.

Next-State Equations

\[ a^* = f(A, B, s) = B + s \cdot A \]

\[ b^* = g(A, B, s) = s + B \cdot A \]

State Next State \( a^* b^* \)

<table>
<thead>
<tr>
<th>State</th>
<th>Next State ( a^* b^* )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

Path for equal delays.

Gates #1 and #2 are slower. Then A is slower than B

A is slow so next state becomes 01

Gates #3 and #4 are slower. Then B is slower than A

The race is noncritical.

No matter which path you take, you always end in state 11

Asynchronous Circuits: © John Knight Electronics Department, Carleton University April 1, 2006
A “Who Cares” Race.

This Race is in state 01 as \( s \) changes. The stable states and all the transitions between them are shown. The “Who Cares” race is between unstable states. Further, no useful transition goes through it. The circuit does not use the transition. No one cares if it has a race.

FIG. 2-23 A Circuit With a “Who Cares” Race.

\[
a^+ = A + s \cdot B + s \cdot B
\]
\[
b^+ = s \cdot B
\]

State Next State
\[
\begin{array}{c|c|c|c}
A & B & s=0 & s=1 \\
00 & 00 & 01 & 00 \\
01 & 00 & 10 & 10 \\
11 & 00 & 11 & 11 \\
10 & 00 & 10 & 10 \\
\end{array}
\]

A Race Between Transient States, But On a Path Between Stable States.

FIG. 2-23 showed a “who cares race” between unused transient states. This race is between transient states, but should not be ignored.

It is an intermediate transition in the path between two stable states.

\[00 \rightarrow 01 \rightarrow 10\]

The second transition from 01 \( \rightarrow \) 10 has a double bit change. The circuit may end up in the stable state 11.

FIG. 2-24

<table>
<thead>
<tr>
<th>State A B</th>
<th>Next State a^+ b^+</th>
<th>s=0</th>
<th>s=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>01 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01 00</td>
<td>10 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 00</td>
<td>11 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 00</td>
<td>10 10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Races Between Transient States Where One Cares!

A Race Between Transient States, But On a Path Between Stable States.

This race is between transient states, but should not be ignored.

It is an intermediate transition in the path between two stable states.

\[00 \rightarrow 01 \rightarrow 10\]

The second transition from 01 \( \rightarrow \) 10 has a double bit change. The circuit may end up in the stable state 11.
Oscillations (Cycles) In State Tables.

Cycles are oscillations
They are transitions that once entered, never reach a stable state.

In theory they oscillate -
forever,
or until an input change takes them out of the cycle.
In practice, some may quickly fall into a stable state.

The Two Types of Cycles:

1. The oscillation continue forever or until turned off.
   These have:
   - no stable state in the path of the cycle
   - no races in the path.

2. The circuit only oscillates a few cycles before falling into a stable state.
   These have:
   - a stable state in the column,
   - a race in the path.¹

¹ For three or more state variables, also check the race can reach a stable state.

---

**FIG. 2-25** A Cycle Which Will Continue Oscillating Forever.

No Stable States In \( s=1 \) Column

When \( s=1 \), it will oscillate forever.

There are two feedback paths

- The one through gate \#2 oscillates (odd number of inverters).
- The one through gate \#1 is a latch (even number of inverters).

The circuit will:

- oscillate when \( s=1 \), and
- latch the last output value when \( s \to 0 \).

**FIG. 2-26** A Cycle, With Stable States In Column, But No Races, Will Continue Forever.

Stable States In the \( s=1 \) Column Without a Race

From the state table; there is a cycle.
From the circuit; the A loop oscillates.
The B loop stays fixed at \( B=0 \).
The A loop oscillating cannot make \( B=1 \)
The A loop oscillating cannot reach a stable state.
Asynchronous Circuits: 
Asynchronous State-Machines With Several Input Variables.

With only one external input.

Single input asynchronous circuits have two types of races:
1. Races between the state bits.
2. Races between a state variable and an input.
   These are called essential hazards and will not be discussed.

With two or more inputs.

In addition, with multiple inputs, one can have:
3. Races between the two inputs.

Multiple Asynchronous Inputs are Miserable

Races between two inputs cannot be handled by this theory.

Theoreticians usually state:
Nearly simultaneous input changes are not allowed.

Practical designers must force rationally behavior for double changes.
One might:
Design the machine to work properly for all states a race might enter.
Interlock the signals so they cannot change at the same time.
Feeds the multiple input signals separately into a synchronous circuit.

A Two-input State-Table With Races

Double input changes are not allowed.

The State-Table has 4 other races.
1. Start at \( \begin{array}{c} 11 \\ \end{array} = \begin{array}{c} 10 \rightarrow s_x = 11. \end{array} 
   This race leads into a cycle.

2. Start at \( \begin{array}{c} 01 \\ \end{array} = \begin{array}{c} 00 \rightarrow s_x = 10. 
   This wraps around the table.
   The final state must be stable.
   It may be any of \( \begin{array}{c} 00, \begin{array}{c} 11, \begin{array}{c} 10.\end{array} \end{array} \end{array} \)

FIG. 2-28

1) \n
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>( a' b' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
</tr>
</tbody>
</table>

No means we don't allow two inputs to change at once.

FIG. 2-27 Two input async. circuit

<table>
<thead>
<tr>
<th>State</th>
<th>( A )</th>
<th>( B )</th>
<th>( a' b' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_x = 00 )</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>( s_x = 01 )</td>
<td>01</td>
<td>01</td>
<td>00</td>
</tr>
<tr>
<td>( s_x = 11 )</td>
<td>11</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>( s_x = 10 )</td>
<td>10</td>
<td>10</td>
<td>00</td>
</tr>
</tbody>
</table>

Races between two inputs cannot be handled by this theory.

Theoreticians usually state:
Nearly simultaneous input changes are not allowed.

Practical designers must force rationally behavior for double changes.
One might:
Design the machine to work properly for all states a race might enter.
Interlock the signals so they cannot change at the same time.
Feeds the multiple input signals separately into a synchronous circuit.
Asynchronous Circuits: Asynchronous State-Machines With

A Two-input State-Table With Races (Cont)

Double input changes are not allowed.
The races continued:

3. Start at (10) with sx=10 -> sx=11.
   No race on first hop
   2nd hop has a race, and
   leads into the same cycle as 1)

4. Start at (11) with sx=10 -> sx=00.
   This may end in either states
   (00) or (01)

FIG. 2-30

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>a</th>
<th>b</th>
<th>0</th>
<th>1</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>10</td>
<td>00</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>00</td>
<td>10</td>
<td>01</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
<td>11</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

5. State variable races with double input variable changes.
   State marked (01)
   The double input change will make a mess of the circuit function.
   We take steps to make sure it will never happen (not allowed).
   Or (next year) follow all possibilities

FIG. 2-29 Two input async. circuit

<table>
<thead>
<tr>
<th>State</th>
<th>Next State a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>sx=00</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>10</td>
</tr>
</tbody>
</table>

Removal of Races

Race-free designs can be done by careful state assignment.
A race is a double-bit state-variable change.
Thus states connected to another state,
must differ in value by one-and-only-one bit
In the Karnaugh map, adjacent squares differ by only one bit.
States can be assigned by plotting them on the Karnaugh map.

FIG. 2-31 Assigning Bit-Patterns to States Using a Karnaugh Map.

Adjacent Karnaugh map squares differ by one bit.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>000</td>
<td>001</td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>010</td>
<td>011</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>110</td>
<td>111</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>

Read the state assignment off the map

Print the state names on map

A = 000
B = 001
C = 011
D = 010
E = 101

Connected states must differ by only one bit
Asynchronous Circuits: © John Knight Electronics Department, Carleton University April 1, 2006

FIG. 2-32 Assigning Bit-Patterns to Difficult State Graphs By Adding States.

A difficult state graph

Here A and R could not be made adjacent because of the triangle.

Triangular states connections like ADR, can never all differ by only one bit

The new state assignment without races

Adding extra state variables will eventually solve the race problem. However it may make the circuit larger and slightly slower.

FIG. 2-33 Assigning Bit-Patterns to States Using Flow-Through States.

A difficult state graph

Since x,y=1,1 sends both R and D to A, let R flow through D. (D acts as a transient state)

Adding extra state variables will eventually solve the race problem. However it may make the circuit larger and slightly slower.

Asynchronous Circuits: © John Knight Electronics Department, Carleton University April 1, 2006

Find A Race Free Assignment for the State Table

(1) Circle stable states.

(2) Fill in the allowed (single input change) transitions from A to other stable states.

(3) Fill in the other transitions between stable states, for single column input changes.
Finding A Race Free Assignment

(3) Sketch transitions on a state table. Try to make transitions horizontal or vertical.

(4) Get rid of angles and jumps

(5) Get rid of 3 sided loops.

(6) Move state locations to map

(7) Rotate map map to place R in 000

Revise some next states to be flow-through states (two hops) to avoid races.

Race Free Assignments

(9) Fill in new transient states and race-free assignment.

(8) Get race-free state assignment off map

\[
\begin{array}{c|c|c|c|c|c|c}
\text{State} & \text{Input } XY=00 & \text{Input } XY=01 & \text{Input } XY=11 & \text{Input } XY=10 \\
\hline
R & R & A & R & B \\
B & A & -- & -- & B \\
C & A & C & R & -- \\
D & D & C & R & D \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{State} & \text{Input } XY=00 & \text{Input } XY=01 & \text{Input } XY=11 & \text{Input } XY=10 \\
\hline
R & R & A & R & B \\
B & A & -- & -- & B \\
C & A & C & R & -- \\
D & D & C & R & D \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\text{State} & \text{Input } XY=00 & \text{Input } XY=01 & \text{Input } XY=11 & \text{Input } XY=10 \\
\hline
R & 000 & R & 000 & A & 001 \\
A & 001 & A & 001 & A & 001 \\
B & 011 & A & 001 & B & 011 \\
C & 011 & C & 011 & D & 010 \\
D & 010 & D & 010 & C & 011 \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c}
\text{State} & \text{Input } XY=00 & \text{Input } XY=01 & \text{Input } XY=11 & \text{Input } XY=10 \\
\hline
R & 000 & R & 000 & A & 001 \\
A & 001 & A & 001 & A & 001 \\
B & 011 & A & 001 & B & 011 \\
C & 011 & C & 011 & D & 010 \\
D & 010 & D & 010 & C & 011 \\
\hline
\end{array}
\]
Hazards Can Poison Asynchronous Machines

Example: The Hazard in the Transparent D-Latch

The simple form of D-latch (transparent latch) is just a MUX.
The latch has a static-1 hazard when \( D, Q = 1, 1 \).
This glitch can feed back and latch itself.

\[ q^* = DC + \overline{C}Q \]

FIG. 2-34

As the clock falls
The 0 may get latched as \( Q = 0 \)
Even though \( D = 1 \), and \( Q_{old} = 1 \)

To mask the hazard
Keep \( DQ = 11 \) across the change in \( C \)
\[ q^* = DC + \overline{C}Q + DQ \]

Hazard-free latch

Common Hazard-Free D-Latch Circuit

Explanitory note (not for examination purposes)

This D-latch circuit is often used and is hazard free.
Analysis of the transistor circuits will show this is about 30% larger than the latch with the hazard, and smaller than the last circuit with the masking gate.

\[ q^* = DC + (D + \overline{C}) \]

No hazard