Consider this implementation (right) of $\overline{A}D + AB\overline{C}$.

(a) What is the difference between a hazard and a glitch?

A hazard is the potential to produce a glitch, which is a spurious (or illogical) output event due to physical characteristics of the circuit realization.

(b) Give a scenario that results in a glitch, of one exists; or no such scenario exists, say so and explain your reasoning.

As can be seen in this K-map. The minimal SOP has two adjacent cubes that do not overlap, indicating a hazard when $B = 1, C = 0, D = 1$ and input $A$ changes.

$\overline{A}D + AB\overline{C}$

The hazard can be masked by adding a redundant clause, $\overline{A}D + AB\overline{C} + B\overline{C}D$ “bridging” the gap between the covering cubes.

It remains to determine whether and under what conditions a glitch actually occurs. The cube adjacency gives the hazard condition, $B\overline{C}D$, so the hazard in this case is associated with a transition on $A$ when $B = D = 1$ and $C = 0$. The question is whether $1 \rightarrow 0$ transition a $0 \rightarrow 1$ transition, or both, on $A$ result in a glitch. The unit-delay timing analysis below shows that when $A$ changes from 0 to 1 a glitch occurs, but not when $A$ changes from 1 to 0.
\[ Y = V + W \]
\[ V = A \overline{D} \]
\[ W = U \overline{C} \]
\[ U = A \overline{B} \]
2.

(a) Design a three-bit “counter” that generates the sequence 0, 1, 3, 7, 6, 4, 0, ….
Assume that D-flip-flops are used in the implementation.

If you happened write down the sequence in binary form, you might have noticed that the sequencer acts as a shifter, moving each bit to the left. After staring a while, you might also see that the complement of the left-most bit cycles back into the right-most.

A more systematic approach is to construct the next-state table (or write it directly into the K-map) and optimize each next-bit function. Either way, the result is:

\[
\begin{align*}
A' &= \overline{C} \\
B' &= A \\
C' &= B
\end{align*}
\]

This design assumes that the sequencer does not start up in an invalid state (101 or 010). In fact, were it ever to reach an invalid state, it would cycle there forever, \(010 \rightarrow 101 \rightarrow 010 \ldots\), as a consequence of the don’t-care value assignments.

(b) Implement your design using only D-flip-flops, 74LS00 \textit{nand} gates and 74LS04 inverters.

It takes three flip-flops and no gates to implement the sequencer.