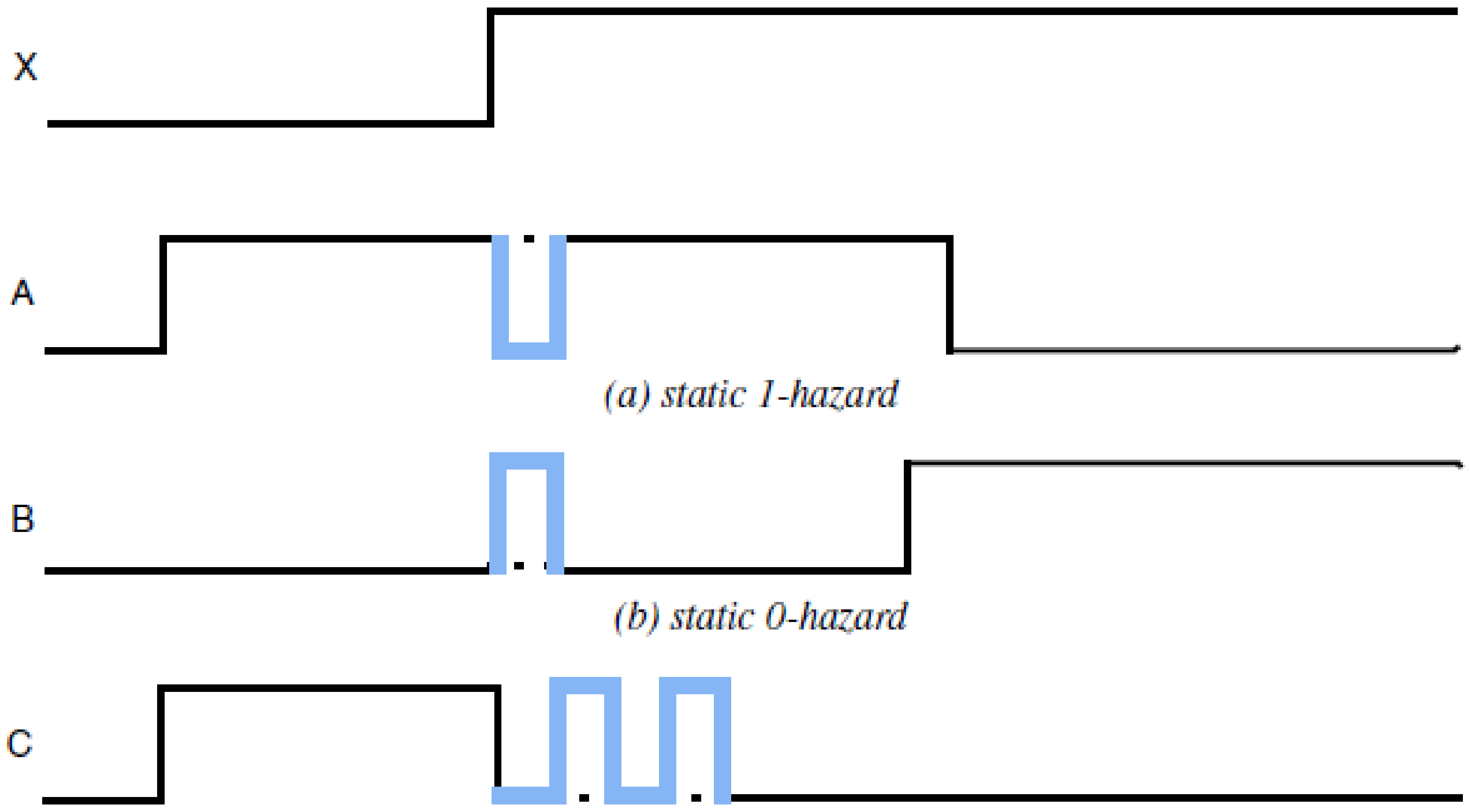
Combinational hazards

- ✓ We break down combinational hazards into two major categories, logic hazards and function hazards.
- ✓ A **logic hazard** is characterized by the fact that it can be eliminated by proper combinational logic design methods.
- ✓ **Function hazards** come with the function being implemented and cannot be dealt with by basic combinational design techniques.

Logic hazards

- ✓ Suppose that input variable changes are spaced such that the effects of a change in one variable is permitted to propagate throughout the circuit before another variable is allowed to change.
- ✓ This is the single-input change case, since input signal patterns can change in only one variable at a time. For example, 00 can be followed by 01 or 10, but not 11.
- ✓ A single-input change static **hazard** (SICS hazard) is a momentary change in an output that occurs as the result of the change of a single input variable when the value of the output variable is to remain fixed.

Single-input change static hazard (SICS hazard)

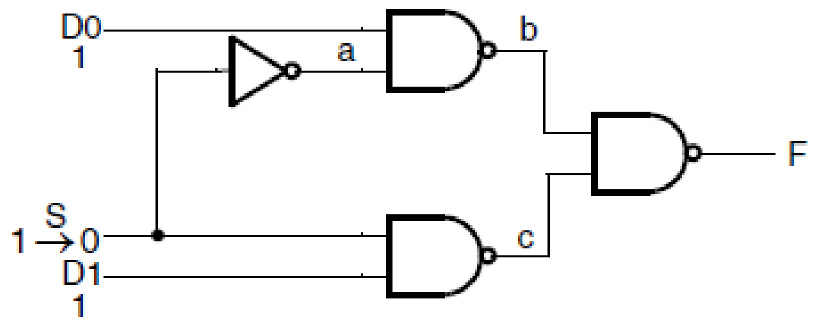


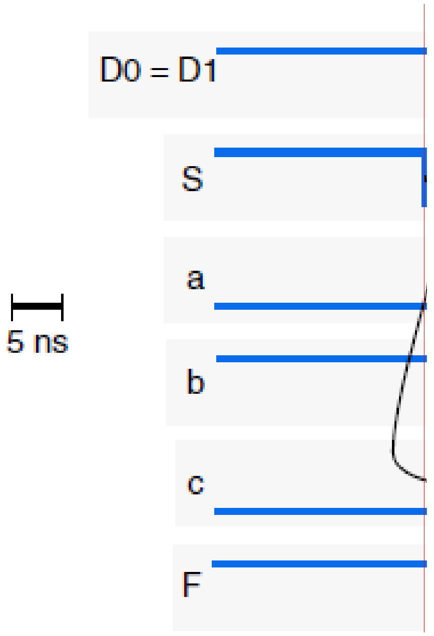
(a) static 1-hazard

(b) static 0-hazard

(c) dynamic hazard

Example of a Static Hazard in a Multiplexer

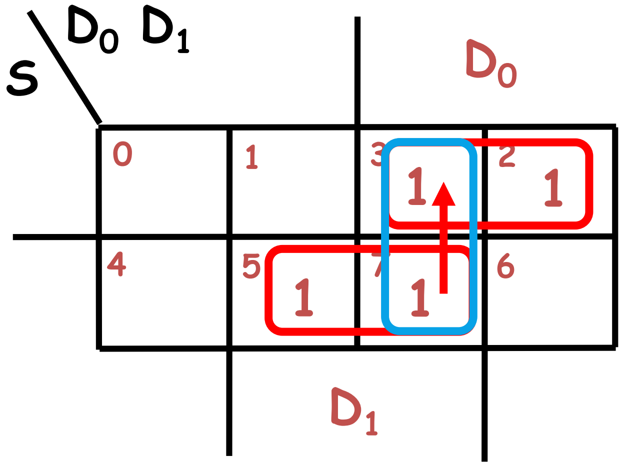


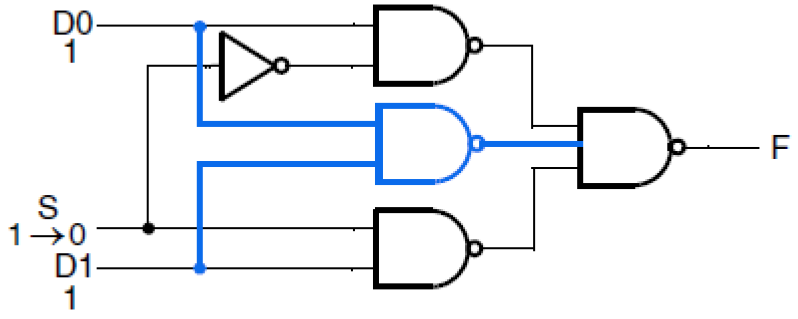


Karnaugh Map of Multiplexer

- ✓ $D_0 = D_1 = 1$ the hazard is in the third column of the map and actually occurs when S goes from 1 to 0
- ✓ The hazard occurs as the circuit goes from minterm $D_0D_1\overline{S}$ to minterm D_0D_1S
- ✓ Combining these two minterms:

  $D_0D_1S + D_0D_1\overline{S} = D_0D_1$
- ✓ The addition of this new term prevents the hazard
- ✓ All of the prime implicants have been used





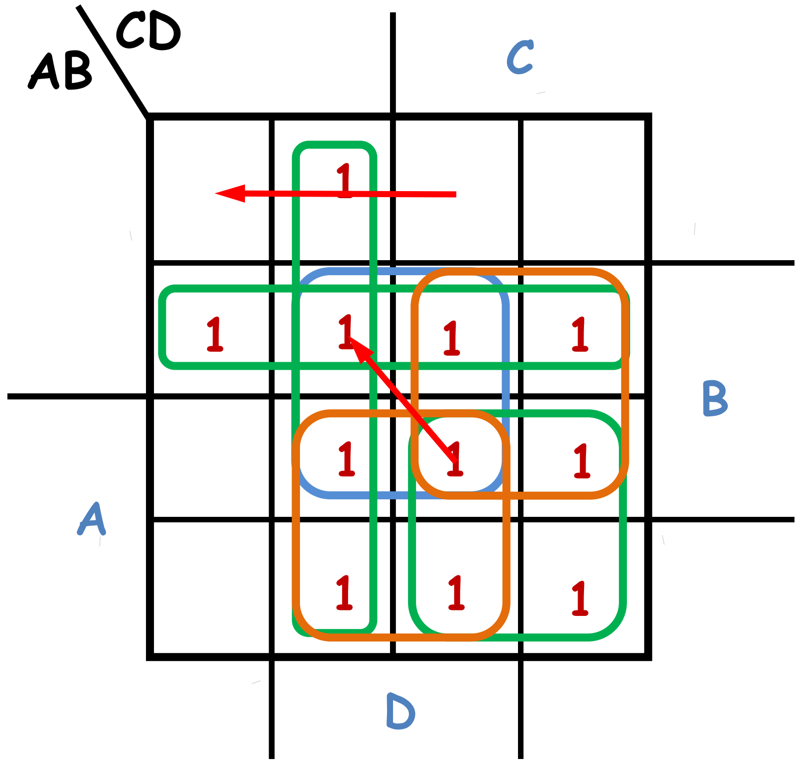
Sum-of-products and static 1-hazards

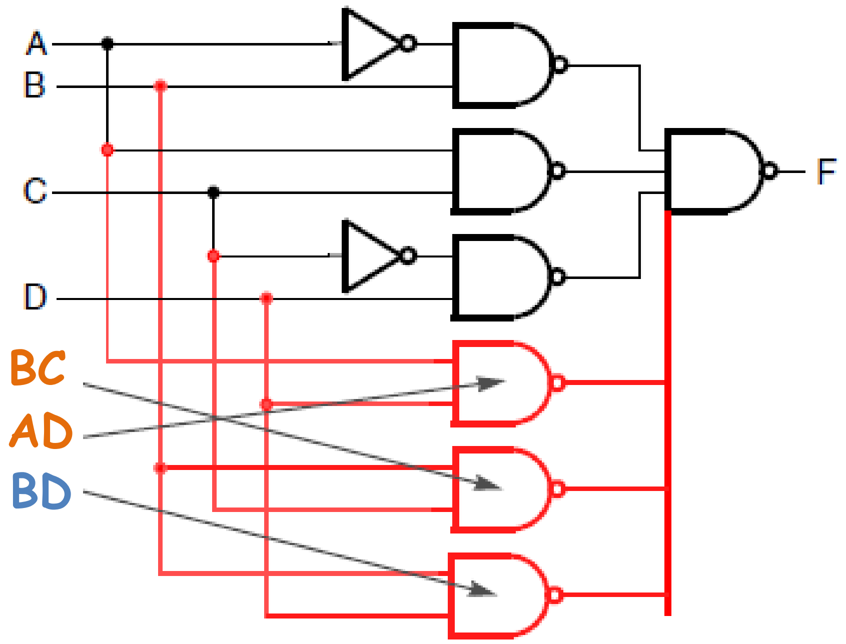
- ✓ In general, for a sum-of-products implementation, the potential for a SICS hazard exists wherever there are two adjacent 1's in the Karnaugh map that are not included within a product term of the implementation.
- ✓ To remove all potential for static 1-hazards from a sum-of-products implementation of *F*, ***all* prime implicants of *F* must be included in the circuit** implementation.
- ✓ A sum-of-products implementation is automatically **free of static 0-hazards**.

Products-of-sum and static 0-hazards

- ✓ For a product-of-sums implementation of function *F*, **all of the prime implicates** of *F* must be included to remove all potential for **static 0-hazards**.
- ✓ ***A* product-of-sums implementation is automatically free of static 1-hazards.**
- ✓ Finally, any sum-of-products or product-of-sums implementation free of static 1-hazards and static 0-hazards is free of dynamic hazards.

Sum-of-products and static 1-hazards





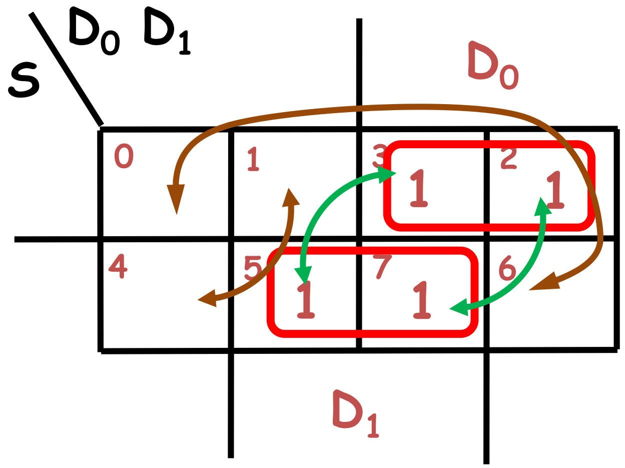
- ✓ The normal minimum solution for *F* is represented by the prime implicants in green.
- ✓ There are three other prime implicants available. The two *A D* and *B C*, are present to deal with single-input change static hazards.
- ✓ The blue, *B D*, is present to deal with a *multiple input- change static hazard* (*MICS hazard*)

Function hazard

- ✓ If multiple variable values can change before that the effect of the first change has not propagated throughout the circuit the elimination of static hazards is not sufficient to guarantee correct operation.
- ✓ For example, consider the change from 0011 to 0000 if *C* changes before *D*, the combination 0001 will momentarily appear generating a 0-hazard.
- ✓ This is not a logic hazard, since there is no way avoid it by changing the implementation logic !
- ✓ Since this hazard is built into the function *F* regardless of implementation, it is called a **function hazard**.
- ✓ The only way to control it is changing relative path delays in the circuit.

Finding Function Hazards

- ✓ 1-function hazards occur between (0,1,1) and (1,0,1) and between (1,1,1) and (0,1,0).
- ✓ 0-function hazards occur between (0,0,1) and (1,0,0) and between (1,1,0) and (0,0,0).
- ✓ Suppose that *D0*, *D1* and *S* can change one, two or three at a time, it is tedious, if not impossible, to manipulate the circuit delays in the multiplexer to avoid all of these function hazards including those involving three variable changes.



Modern chip design aspects

- ✓ Modern chips became too complex
- ✓ The number of transistors in a modern chip is over a 100 M
- ✓ Transistor count per chip and chip speed rise up to 50% per year
- ✓ Estimated time needed for manual implementation (100 M transistor, 10 sec/transistor) – **135.5 years**!!!

VHDL

- ✓ **VHDL** - VHSIC Hardware Description Language
- ✓ **VHSIC** - Very High Speed Integrated Circuit
- ✓ Development of VHDL began in 1983, sponsored by Department of defense, further developed by the IEEE and released as IEEE Standard 1076 in 1987
- ✓ Today it is De facto industry standard for hardware description languages

The abstraction hierarchy

- ✓ **The abstraction hierarchy** can be expressed in two domains: structural domain, behavioral domain
- ✓ **Structural domain** – component model is described in terms of an interconnection of more primitive components
- ✓ **Behavioral domain** – component model is described by defining its input/output response
- ✓ **VHDL** is used for both structural and behavioral description
- ✓ Six abstraction hierarchy levels of detail commonly used in design: silicon, circuit, gate, register, chip and system

Design process

- ✓ The design cycle consists of a series of transformations, **synthesis steps**:
  - Transformation from English to an algorithmic representation,
    natural language synthesis
  - Translation from an algorithmic representation to a data flow representation,
    algorithmic synthesis
  - Translation from data flow representation to a structural logic gate representation,
    logic synthesis
  - Translation from logic gate to layout and circuit representation,
    layout synthesis

Design tools

- ✓ **Editors** – textual (circuit level – SPICE gate, register, chip – VHDL) or graphic (used at all levels)
- ✓ **Simulators** – stochastic (system level) or deterministic (all levels above the silicon level)
- ✓ ***Checkers and Analyzers*** – employed at all levels, used for example
  - to insure that the circuit layout can be fabricated reliably (rule checkers),
  - to check for the longest path through a logic circuit or system (timing analyzers)
- ✓ **Synthesizers and Optimizers** – improving a form of the design representation