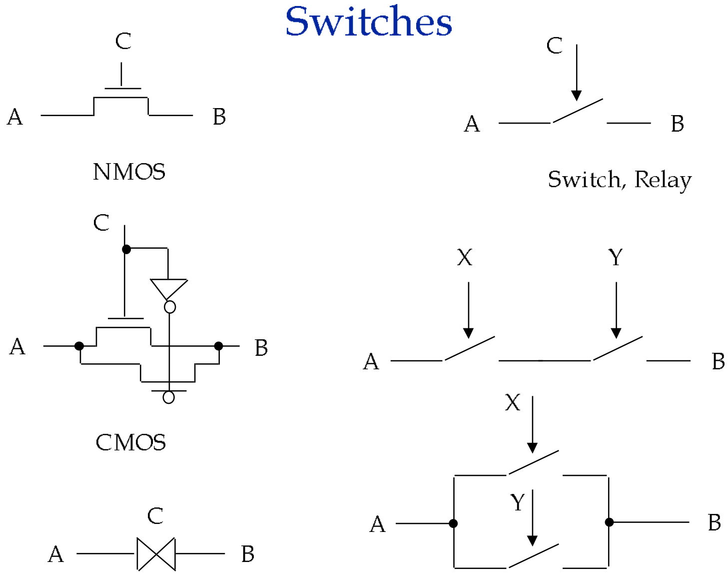
Outline

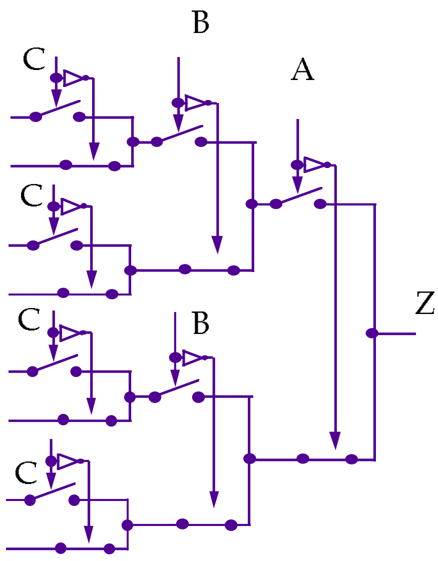
- ❍ Last time:
  - ➜ Implementation of logic functions: TTL, CMOS
  - ➜ Delay models: Transition time, propagation delay
  - ➜ Hazards and "Glitches"
  - ➜ Hazard-free design
  - ➜ Summary of combinational circuit design
- ❍ This lecture:
  - ➜ Switch-level implementation
  - ➜ Design Using Multiplexers
  - ➜ Shannon Expansion
  - ➜ Binary Decision Diagrams (BDDs)

Implementation of Logic Using Switches

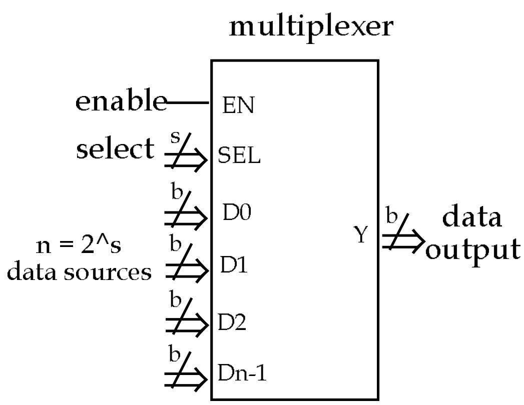


Series-Parallel Networks

Multiplexer Trees Using Switches



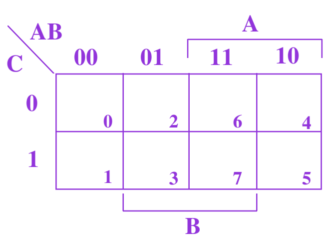
Multiplexer



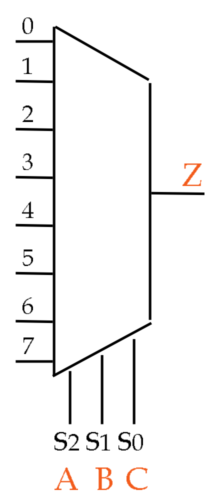
Consider b=1, s=1

Consider b=1, s=2

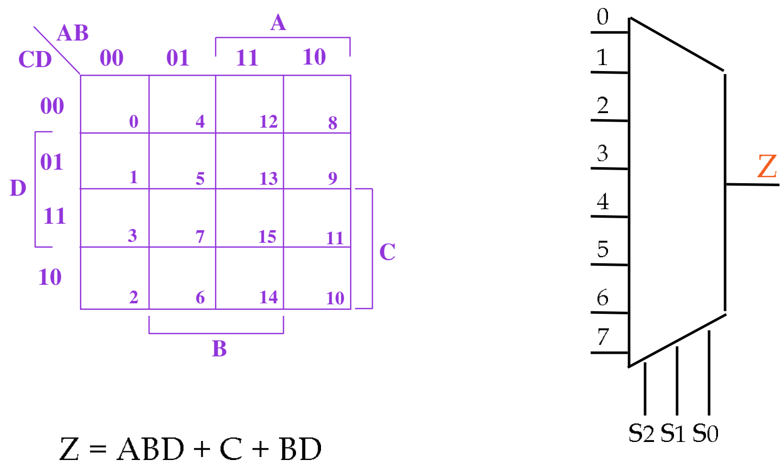
Design Using Multiplexers



$Z = AB + C$



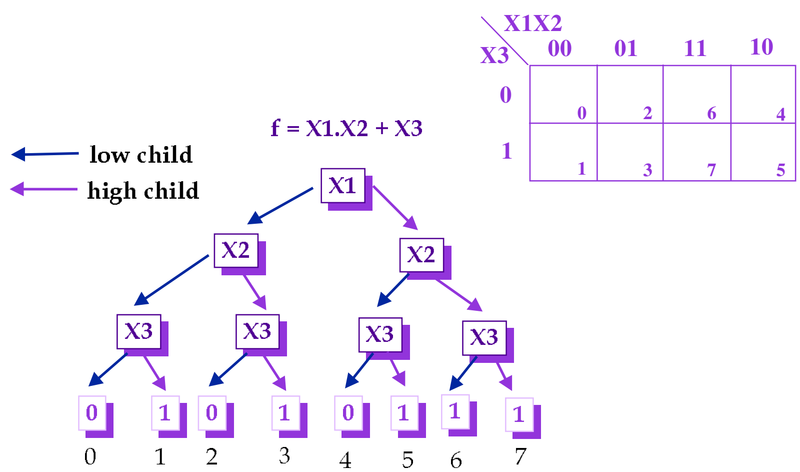
Design Using Multiplexers



Z = ABD + C + BD

Binary Decision Diagrams

❍ **In general: fv = Xv' flow + Xv fhigh**



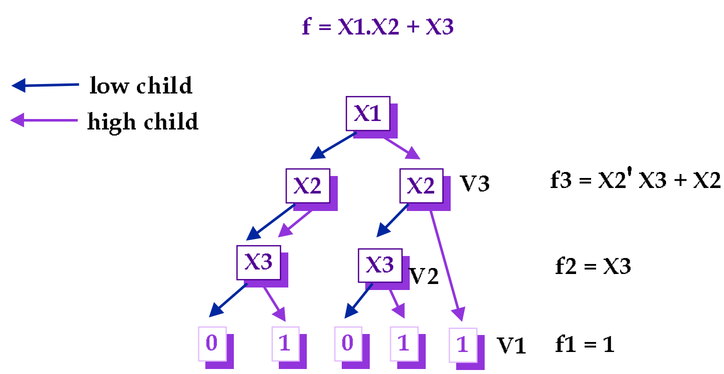
Implementation of Logic Using Switches

Shannon Expansion:

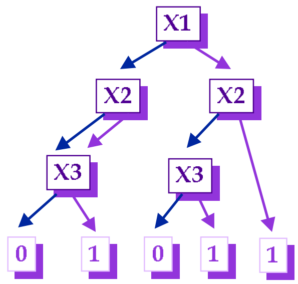
(T15) $F(X,Y,Z) = X \bullet F(1,Y,Z) + X' \bullet F(0,Y,Z)$

(T15') $F(X,Y,Z) = (X+F(0,Y,Z)) \bullet (X'+F(1,Y,Z))$

Binary Decision Diagrams



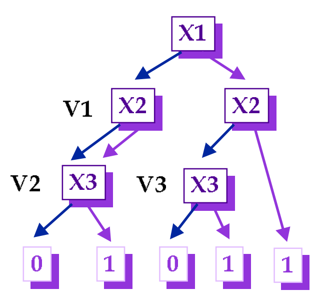
Use of BDDs for Verification



- Condition on edges forces order on variables.
- Order must be consistent with all edges.
- For each edge, parent before child in the order.
  - ➜ X1 X2 X3

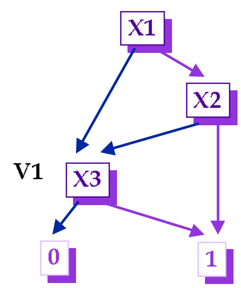
Use of BDDs for Verification

- V1 is a redundant vertex
- V2, V3 represent the same function
- A BDD is a reduced binary decision graph
  - ➜ Reduction is O(N*log(N)) for N verticies

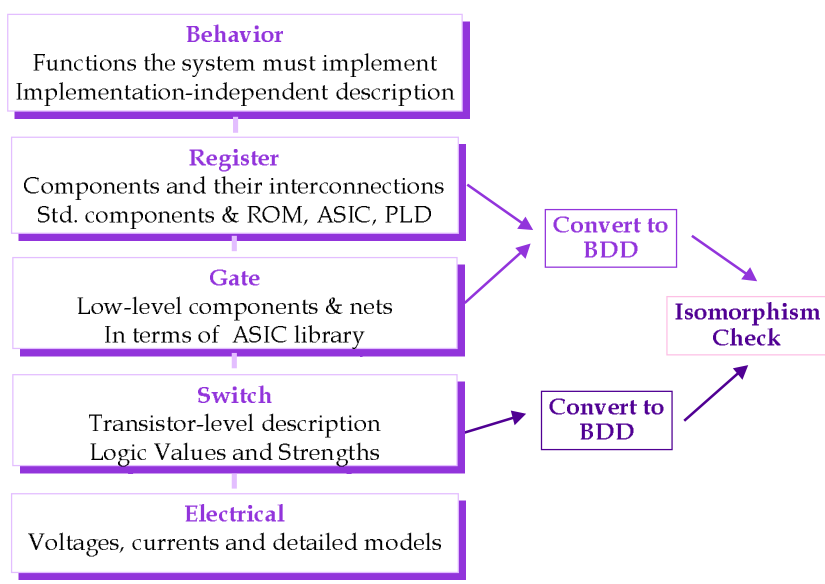


Use of BDDs for Verification

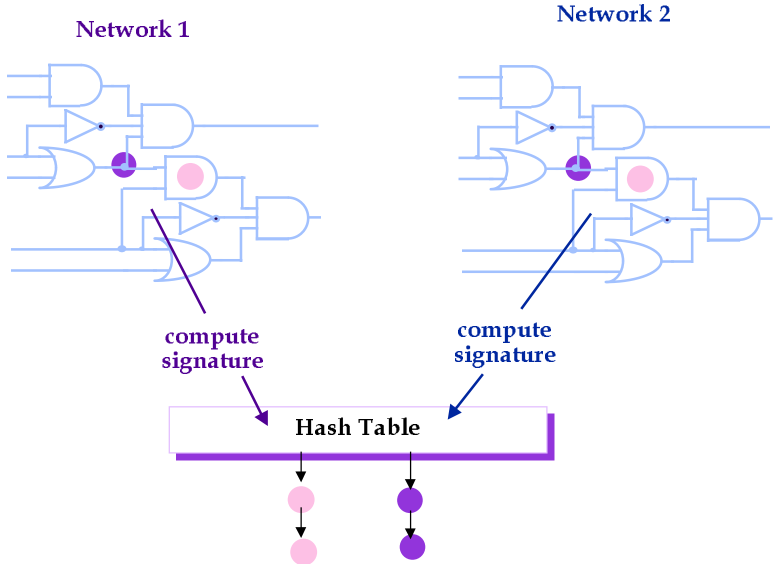
- Each vertex corresponds to a partial assignment of inputs.
  - ➜ V1: 0-, 10
- Result of reduction is a canonical form.



Combinational Verification Using Canonical Form



Connectivity Verification



Connectivity Verification

(1) Read Network1 and Network2 into separate graph data structures (usually, node = transistor or gate, edge = connection).

(2) Compute signatures for nodes or edges or both.

- ➔ **Type-specific: Gate type, #inputs, #outputs**
- ➔ **Network-specific (local): Fanin types, #fanouts, fanout types**
- ➔ **Network-specific (global): Distance from primary inputs, primary outputs, nodes that are known to be the same in each network (e.g. named, primary inputs or outputs ("seeds")).**

(3) Hash signatures from both networks into single hash table.

(4) If a hash table cell has:

- ➔ **> 2 nodes: ignore for now**
- ➔ **= 2 nodes: a match has been found**
- ➔ **= 1 node: an (easy!) error has been found**

(5) Add links between networks for nodes that have been matched.

(6) Recompute hash functions for un-bound nodes and repeat until done or no change.