

VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

Lecture 8.1 Logic Synthesis: Multilevel Logic – Implicit Don't Cares, Part 1



Chris Knott/Digital Vision/Getty Images

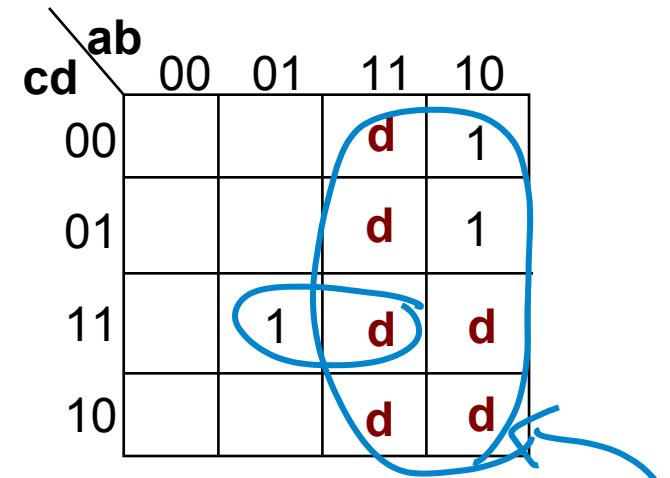
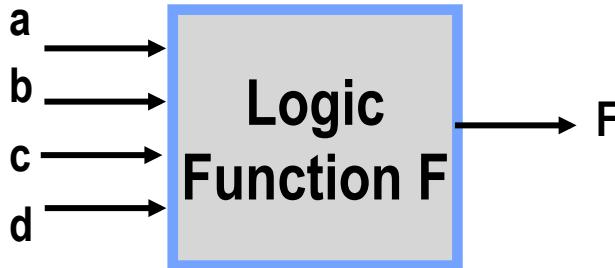
Philosophy

- We made progress on multi-level logic by **simplifying the model**
 - Algebraic model – we got **rid** of a lot of “difficult” Boolean behaviors
 - But we **lost** some optimality (some result quality) in the process
- How do we put it back? One surprising answer: **Don’t cares**
 - We run ESPRESSO-style simplification on 2-level SOP form in each node of network
 - To help this, **extract don’t cares** from “surrounding logic,” use them **inside each node**
- The big difference in multi-level logic
 - Don’t cares happen as a natural byproduct of Boolean network model, called **Implicit**
 - They are all over the place, in fact. Very useful for simplification
 - But they are *not* explicit. **We have to go hunt for them...**



Don't Cares Review: 2-Level

- In basic digital design...
 - Don't Care (DC) = an input pattern that can **never** happen



Patterns $a \ b \ c \ d = 1010, 1011, 1100, 1101, 1110, 1111$ cannot happen
So, we are free to decide if $F=1$ or $F=0$ here, to better **optimize F**



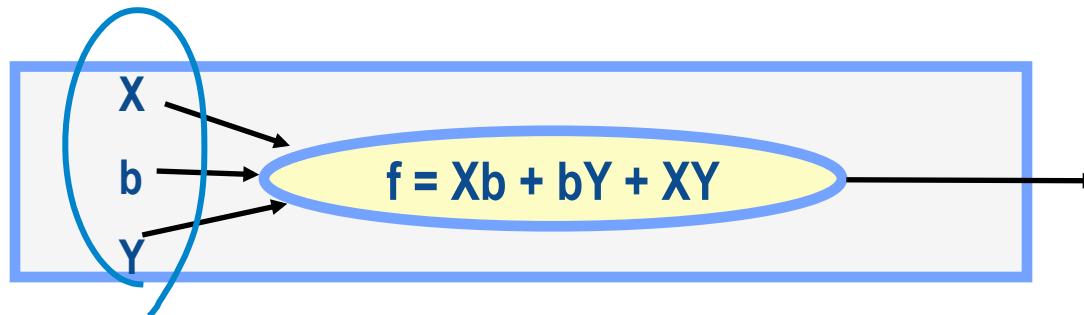
Don't Cares (DCs): Multi-level

- **What's different in multi-level?**
 - DCs arise **implicitly**, as a result of the Boolean logic network structure
 - We must go **find** these implicit don't cares – we must search for them explicitly
- **Nice references for these ideas**
 - Giovanni DeMicheli, *Synthesis and Optimization of Digital Circuits*, McGraw Hill, 1994
 - Some of my development is based on ideas from Chapter 8.
 - Srinivas Devadas, Kurt Keutzer, Abhijit Ghosh, *Logic Synthesis*, McGraw-Hill, 1994
 - The *most* complete paper is: Bartlett, Brayton, Hachtel, Jacoby, Morrison, Rudell, Sangiovanni-Vincentelli, Wang, "Multi-Level Logic Minimization Using Implicit Don't Cares," *IEEE Transactions of CAD*, vol. 7, no. 6, June 1988, pp 723-740 .



Multi-level DCs: Informal Tour

- Suppose we have a Boolean network: lets look at node “f”
- Can we say anything about don’t cares for node f?
 - **No:** we don’t know any “context” for surrounding parts of network
 - As far as we can tell, all patterns of inputs (X, b, Y) are **possible**



A Karnaugh map for the function $f = Xb + bY + XY$. The columns are labeled Xb and Y , and the rows are labeled 0 and 1. The map shows the following values:

	00	01	11	10
0	0	0	1	0
1	1	1	1	1

Two specific cells in the bottom row are circled in blue: the cell for input pattern $(Xb=1, Y=1)$ and the cell for $(Xb=1, Y=0)$.



Multi-level DCs: Informal Tour

- OK, suppose we know this about input X to f

- Node X is actually $a \cdot b$ – now can we say something about DCs for node f ...?

- YES

PIs

a

b

$X = ab$

There are now some
impossible patterns
of variables (X, b, Y) !

b

$f = Xb + bY + XY$

POs

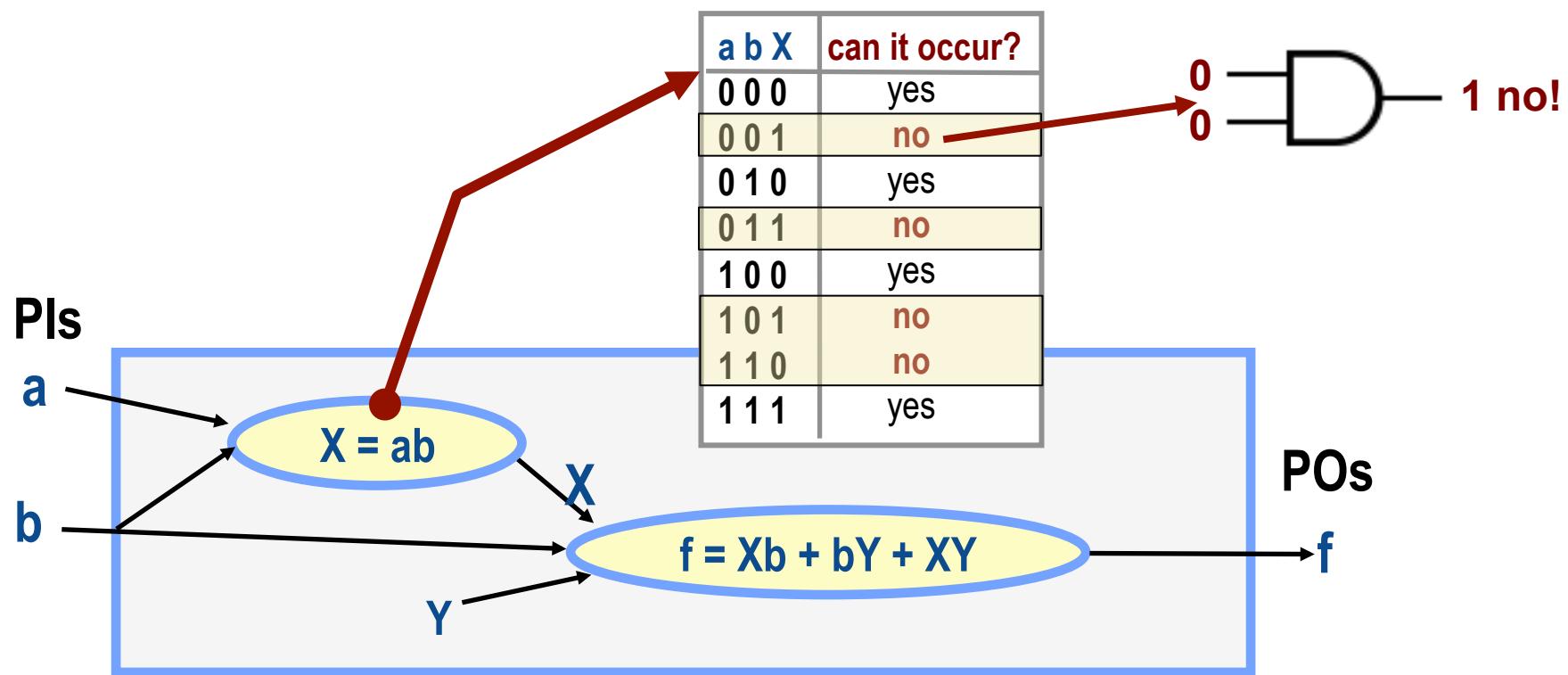
f

(To be clear: **PIs** = Primary Inputs; **POs** = Primary Outputs)



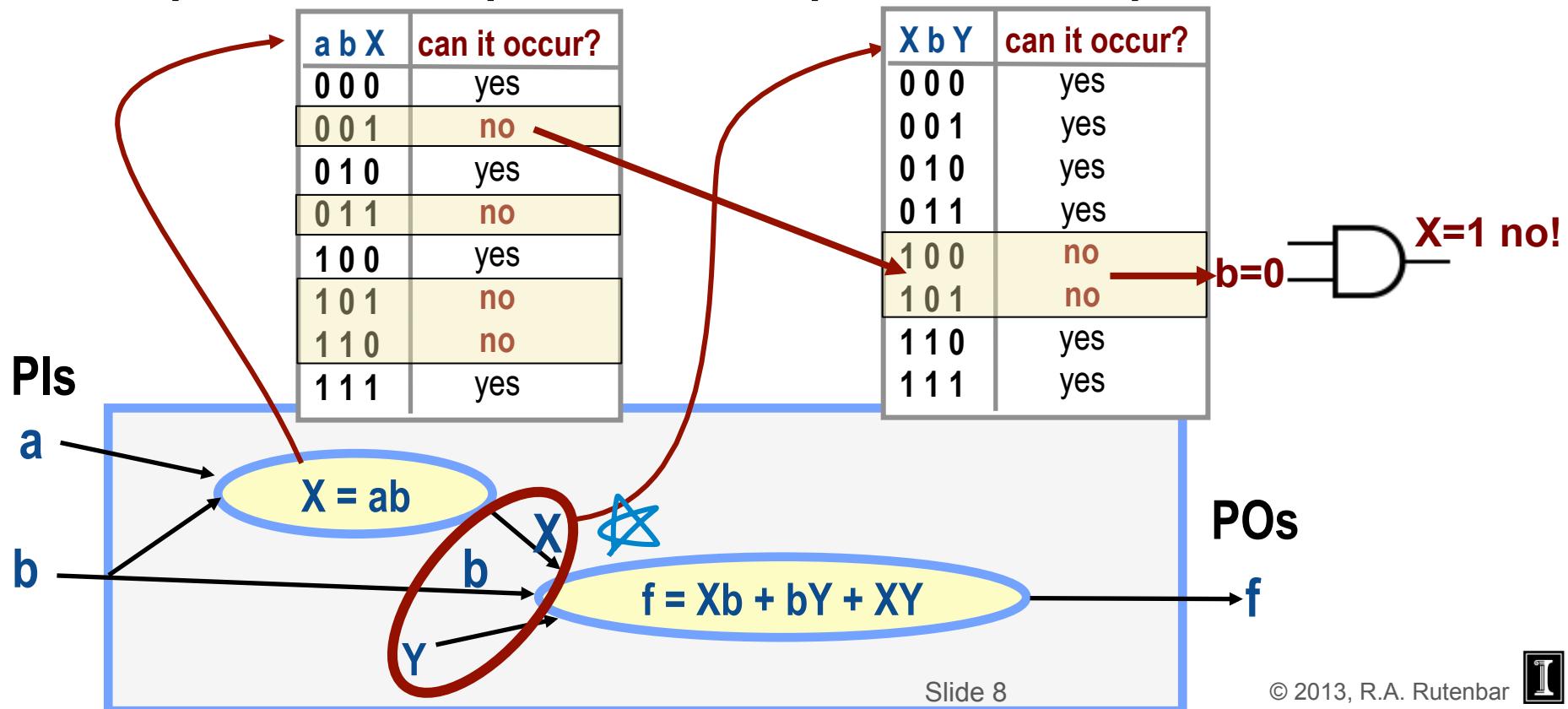
Multi-level DCs: Informal Tour

- Go list all the input/output patterns for node X



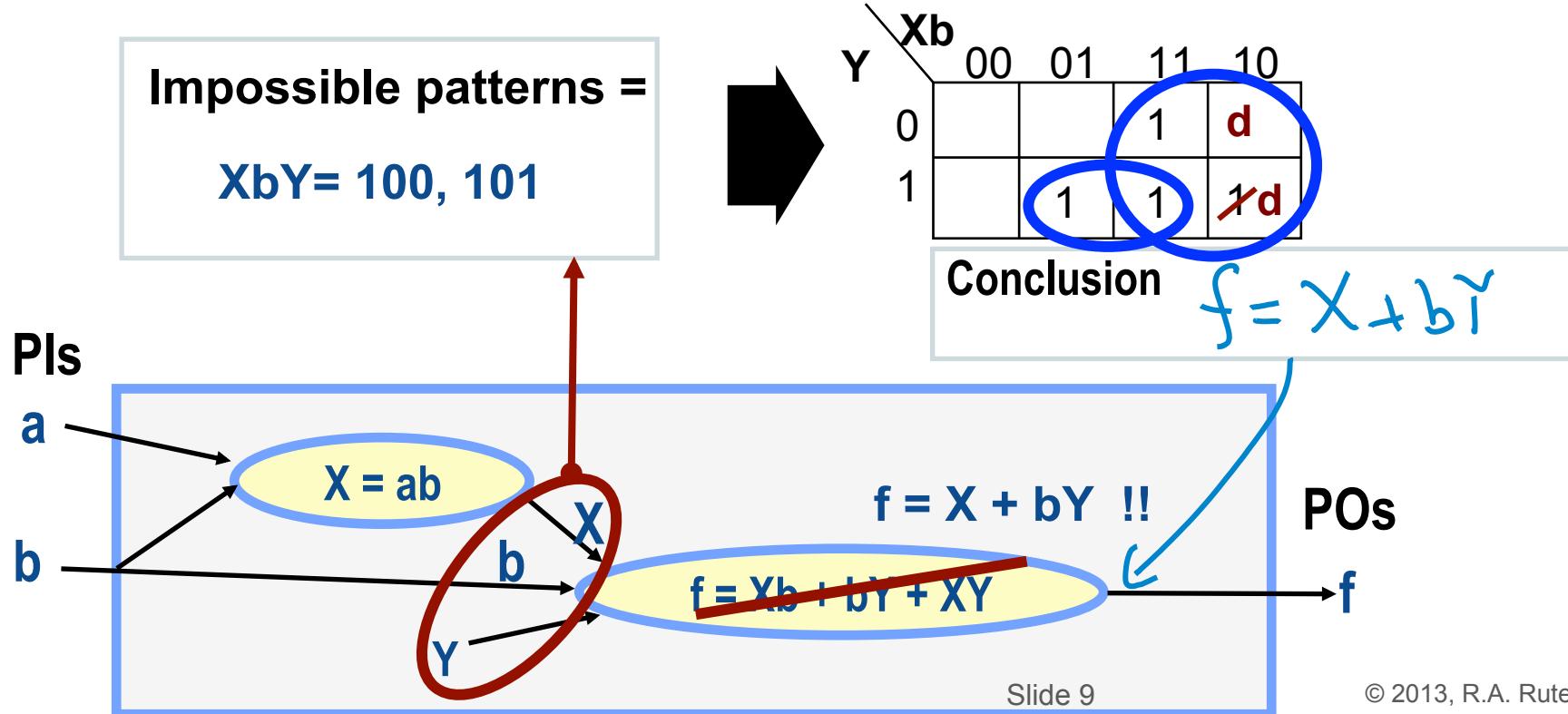
Multi-level DCs: Informal Tour

- Impossible $a b X$ patterns \rightarrow Impossible $X b Y$ patterns?



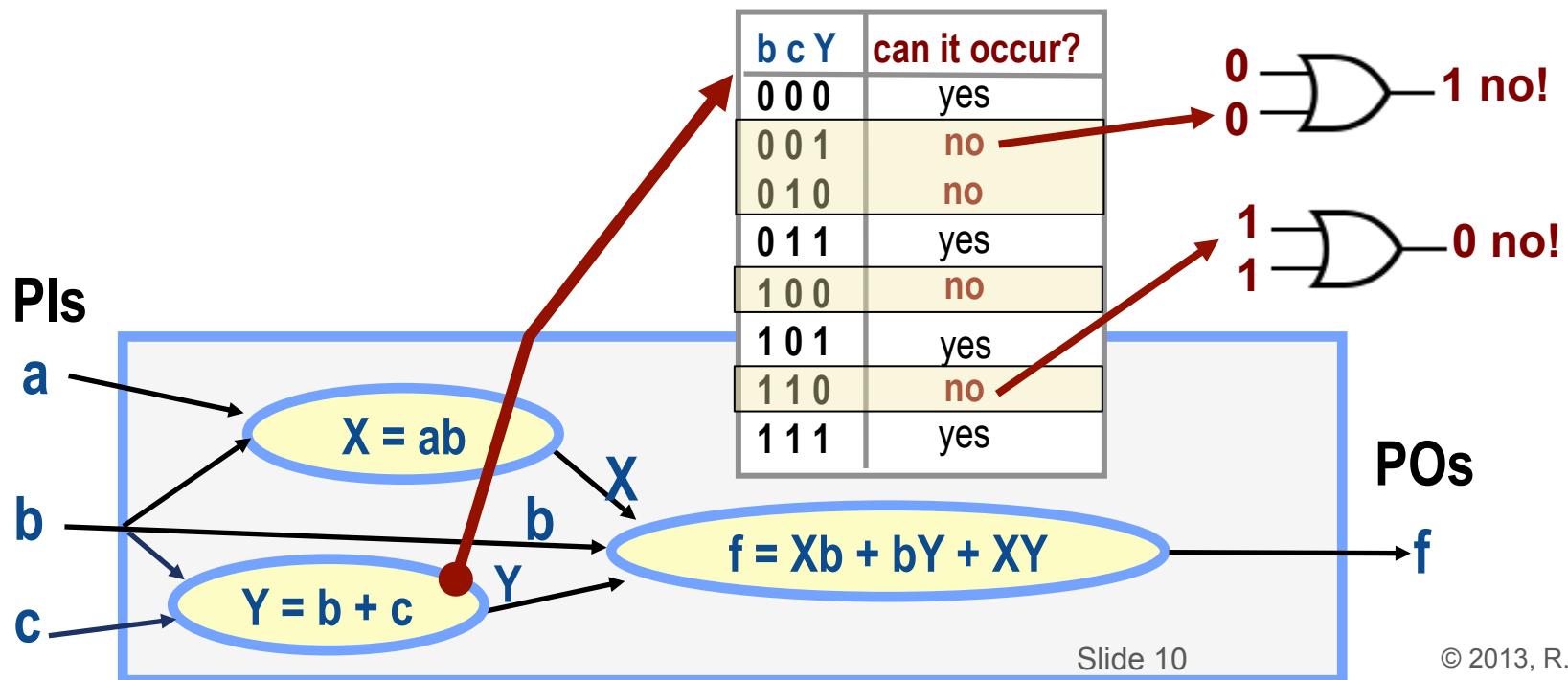
Impossible X b Y Patterns Give DCs for Node f

- These impossible $X b Y$ patterns change how we can simplify f



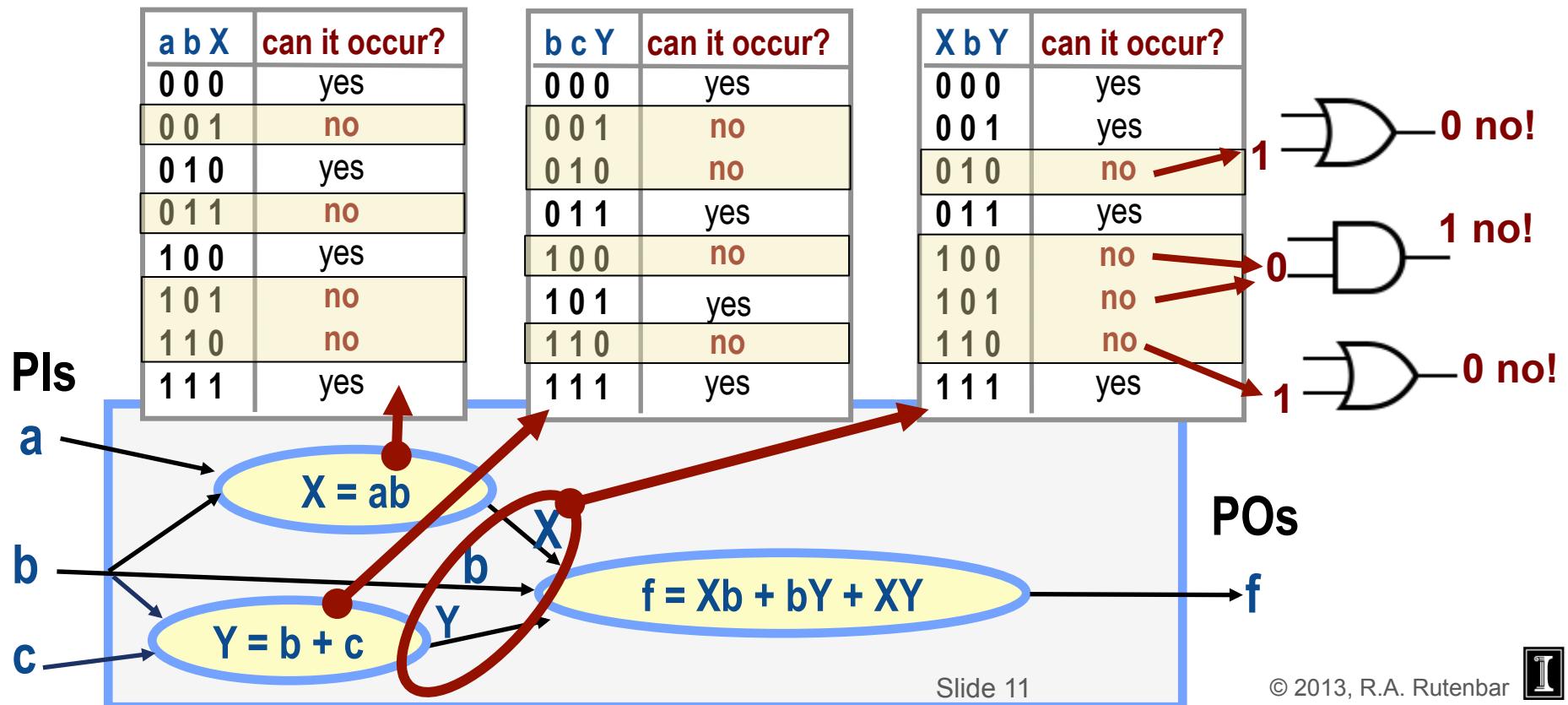
Back to Our Multi-level DC Tour...

- OK, what if we now know $Y = b+c$ as well?
- Can do this again at Y ...are there impossible patterns of $b\ c\ Y$?



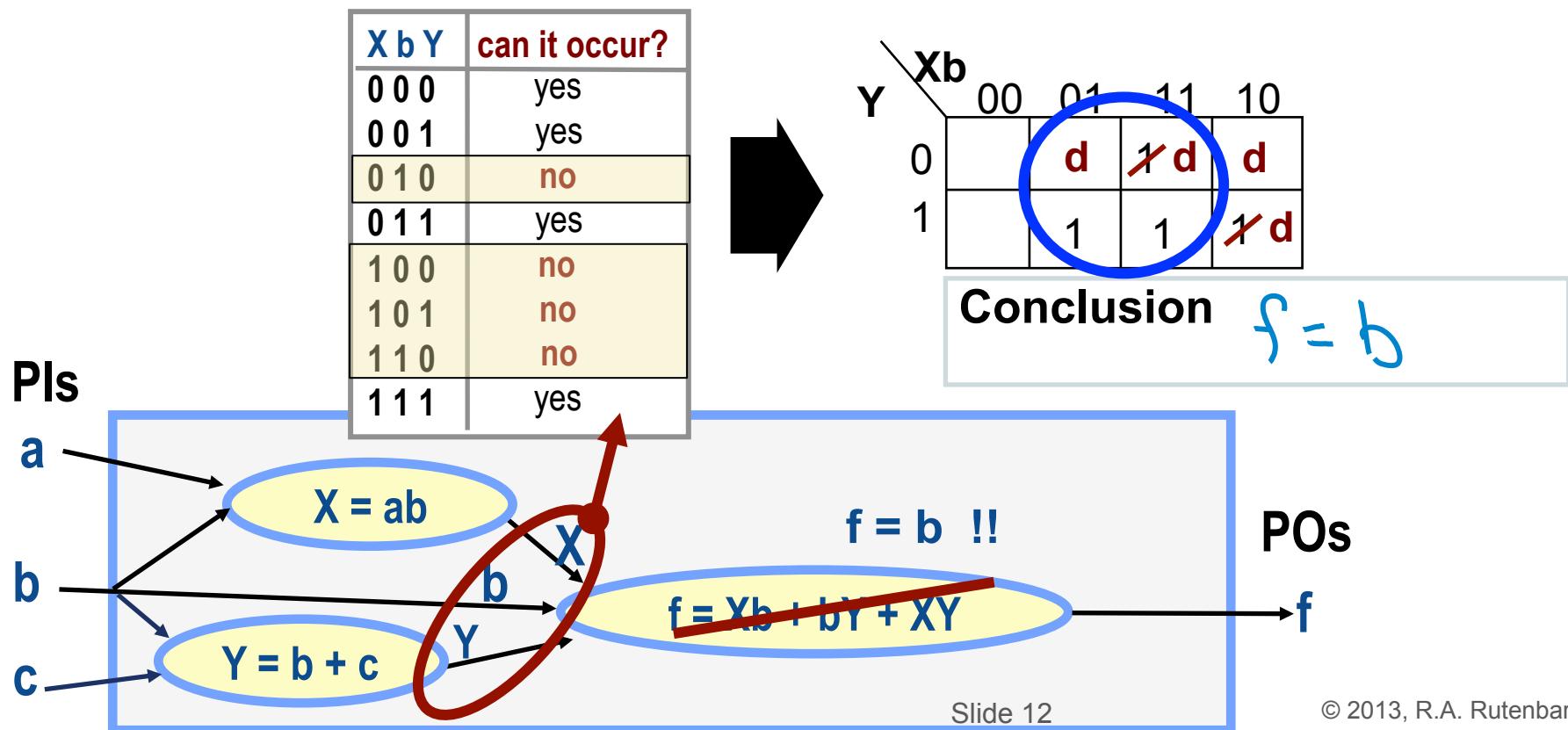
From Impossible Patterns of $X, Y \rightarrow f$

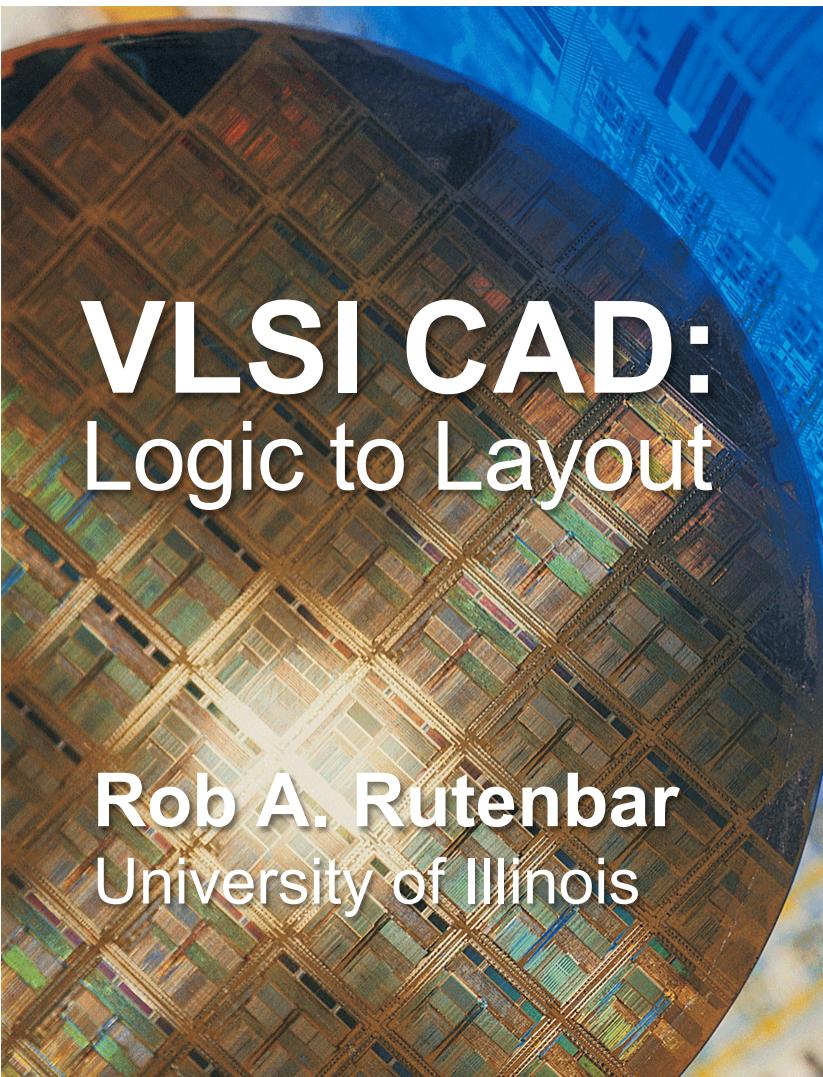
- So, can we (again) get impossible patterns on $X b Y$?



Informal Multi-level DC Tour

- So, do these change how we can simplify inside f ?





VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

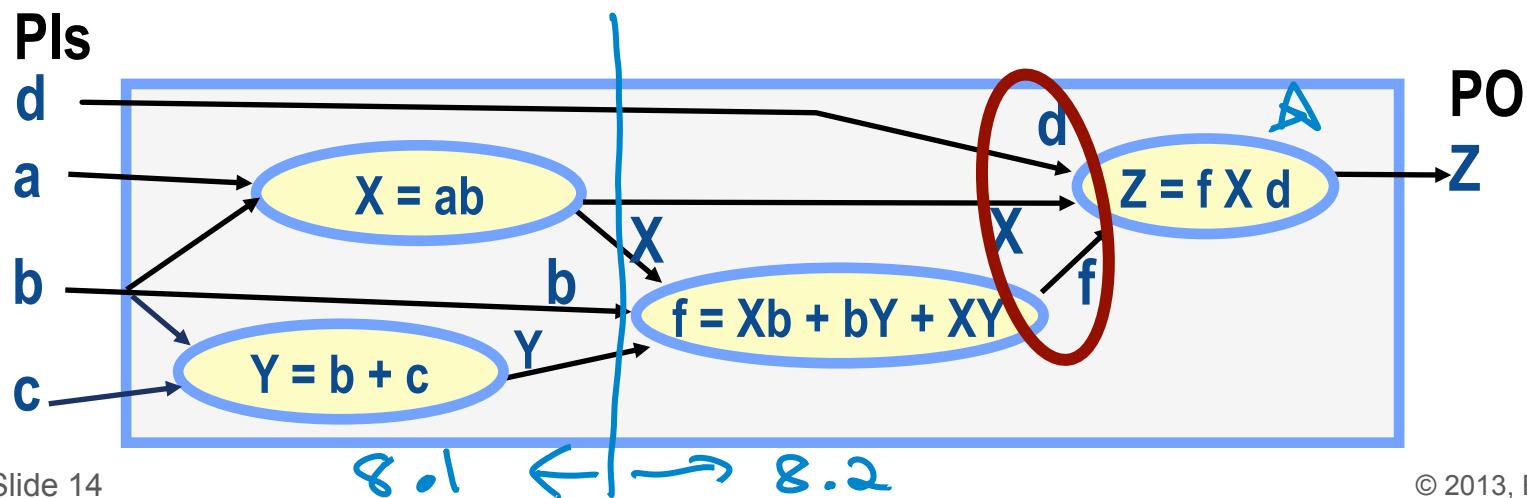
Lecture 8.2 Logic Synthesis: Multilevel Logic – Implicit Don't Cares, Part 2



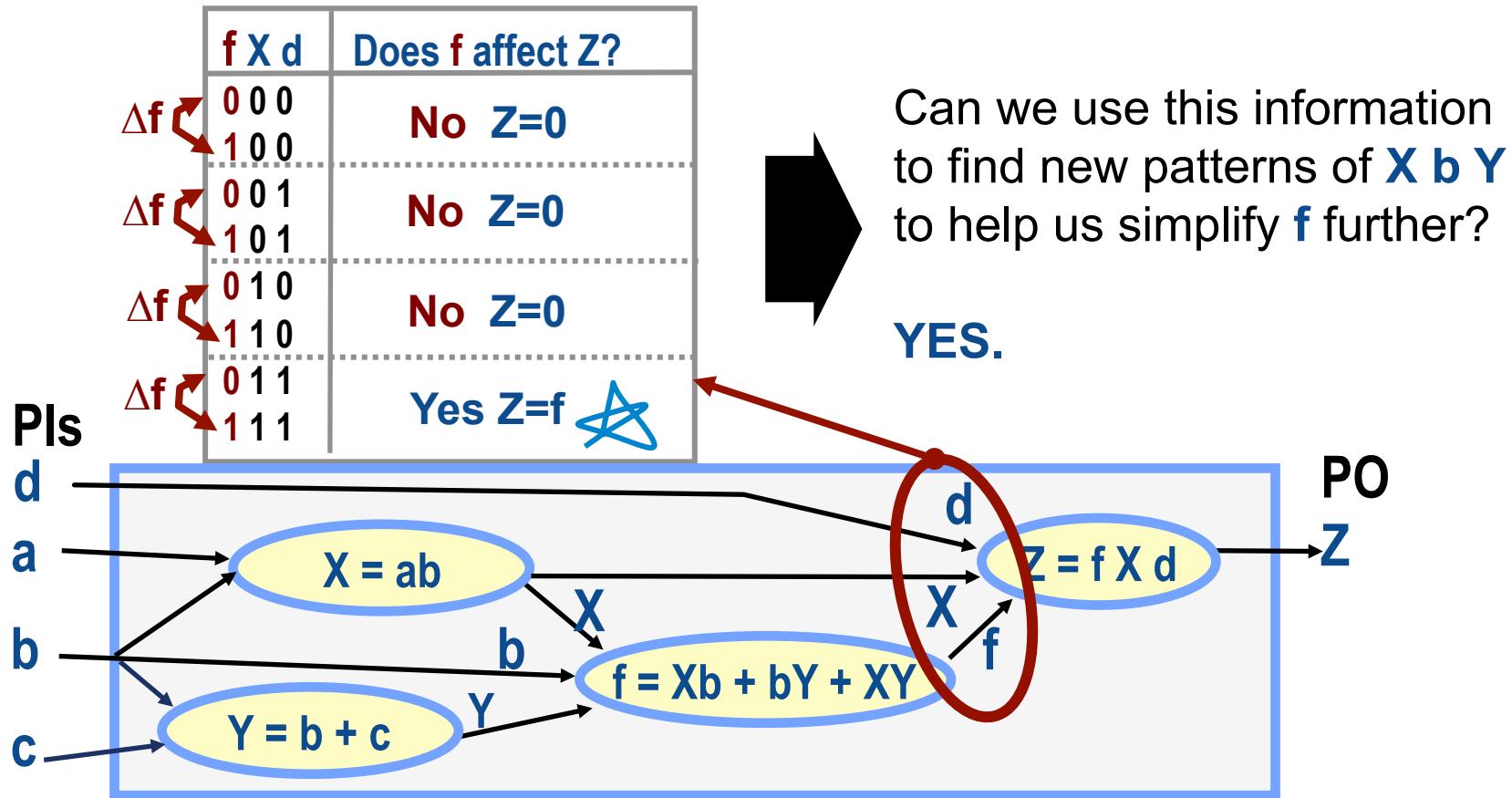
Chris Knott/Digital Vision/Getty Images

Continue Our Multi-level DC Tour...

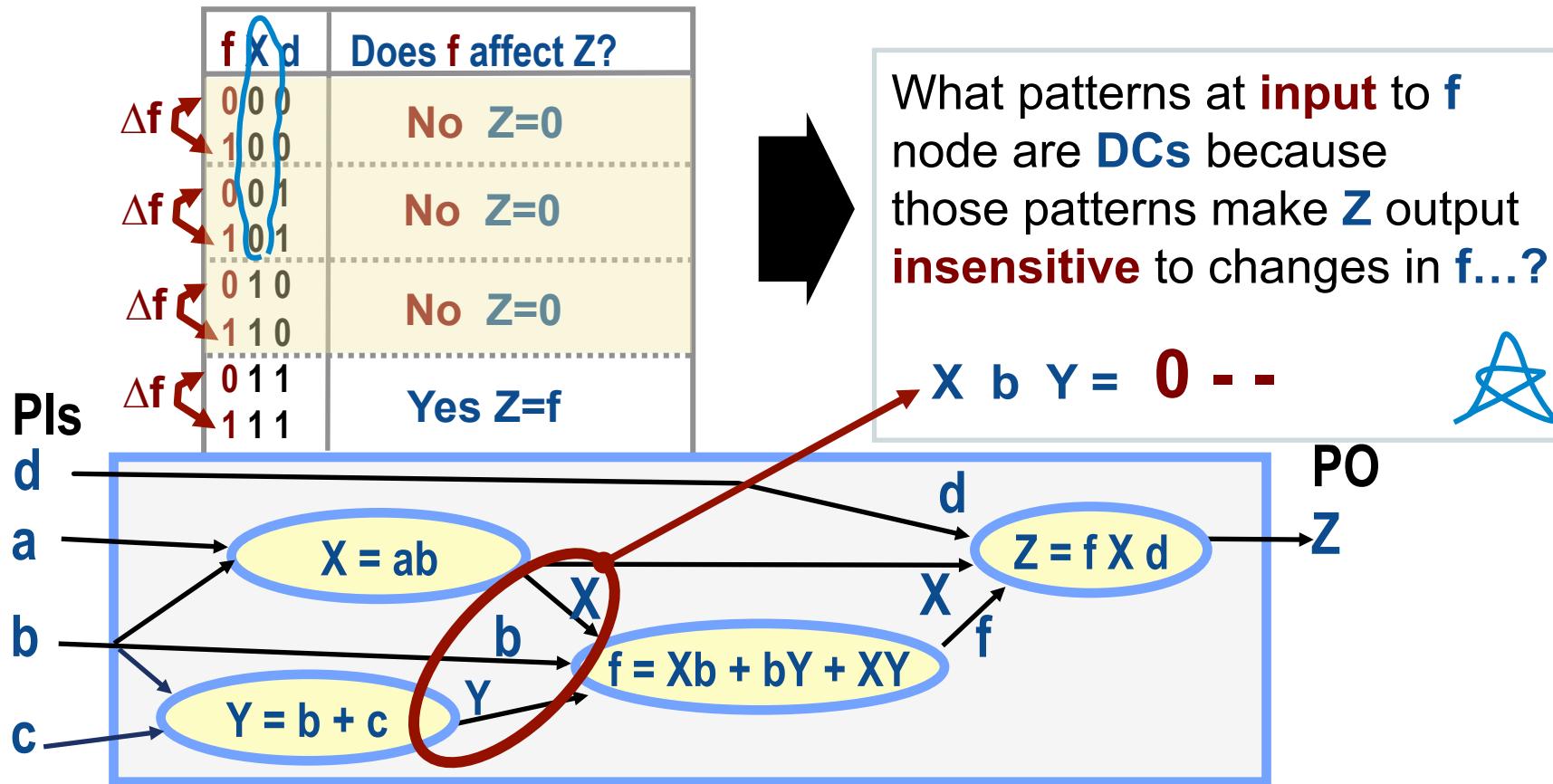
- OK, now suppose f is not a primary output, Z is...
 - Question: when does the value of the output of node f actually affect the network primary output Z ...?
 - Or, said conversely: When does it not matter what f is...?
 - Let's go look at patterns of $f \times d$ at node Z ...



When Is Z “Sensitive” to Value of f



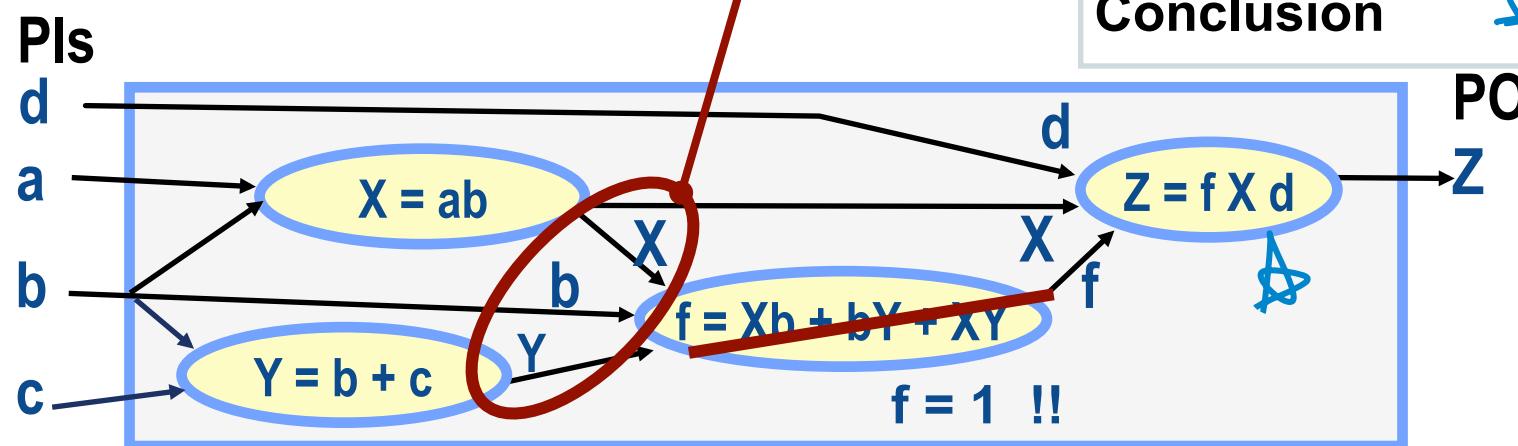
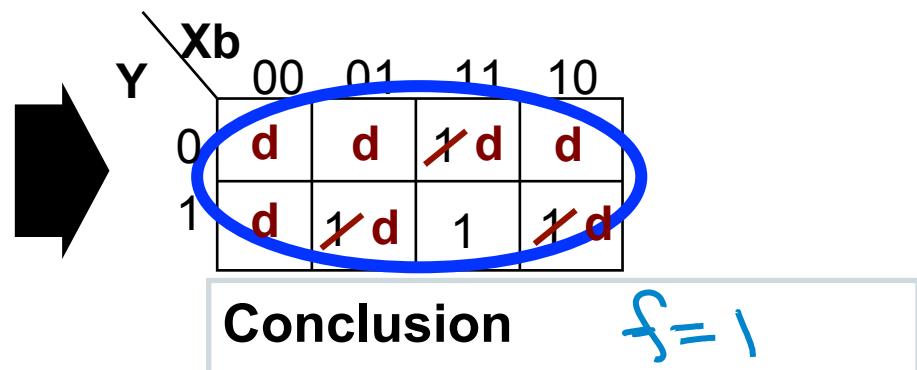
When Is Z “Sensitive” to Value of f



Back to Multi-level DC Tour

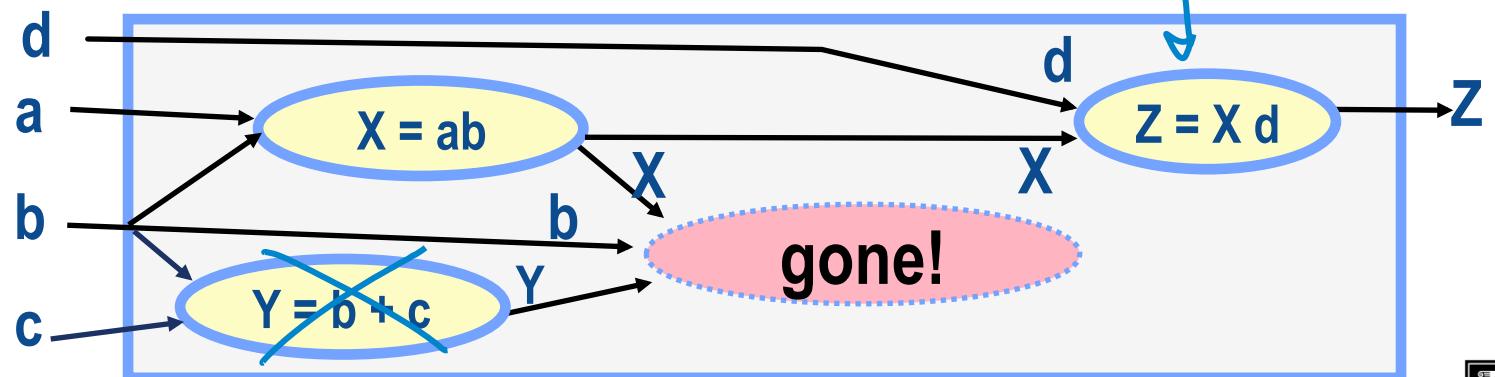
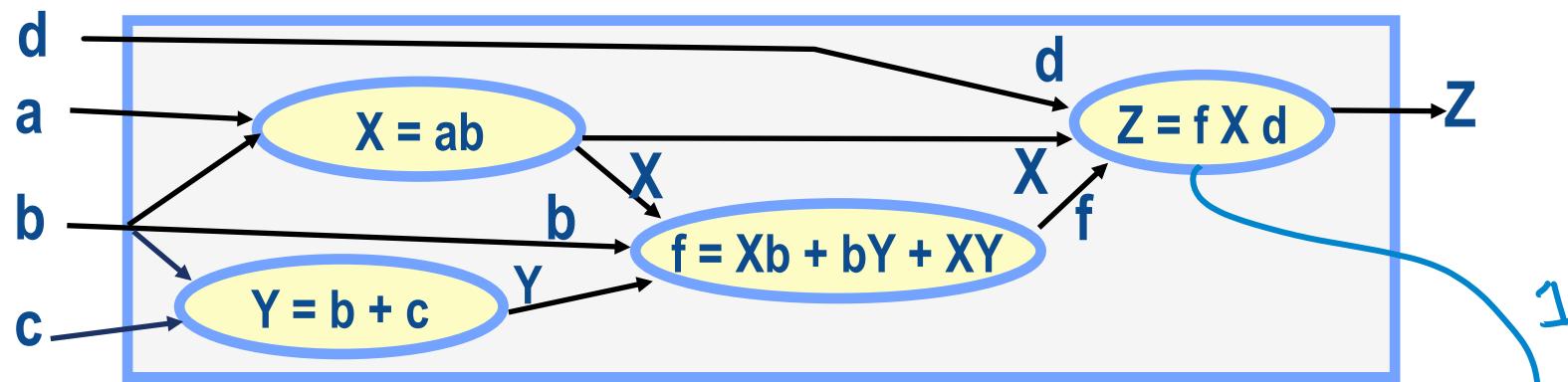
- So, can we use this new $X=0$ DC pattern to simplify f more?

Pattern $XbY = 0--$ at **input** to f
will make network's Z output
insensitive to changes in f



Final Result: Multi-level DC Tour

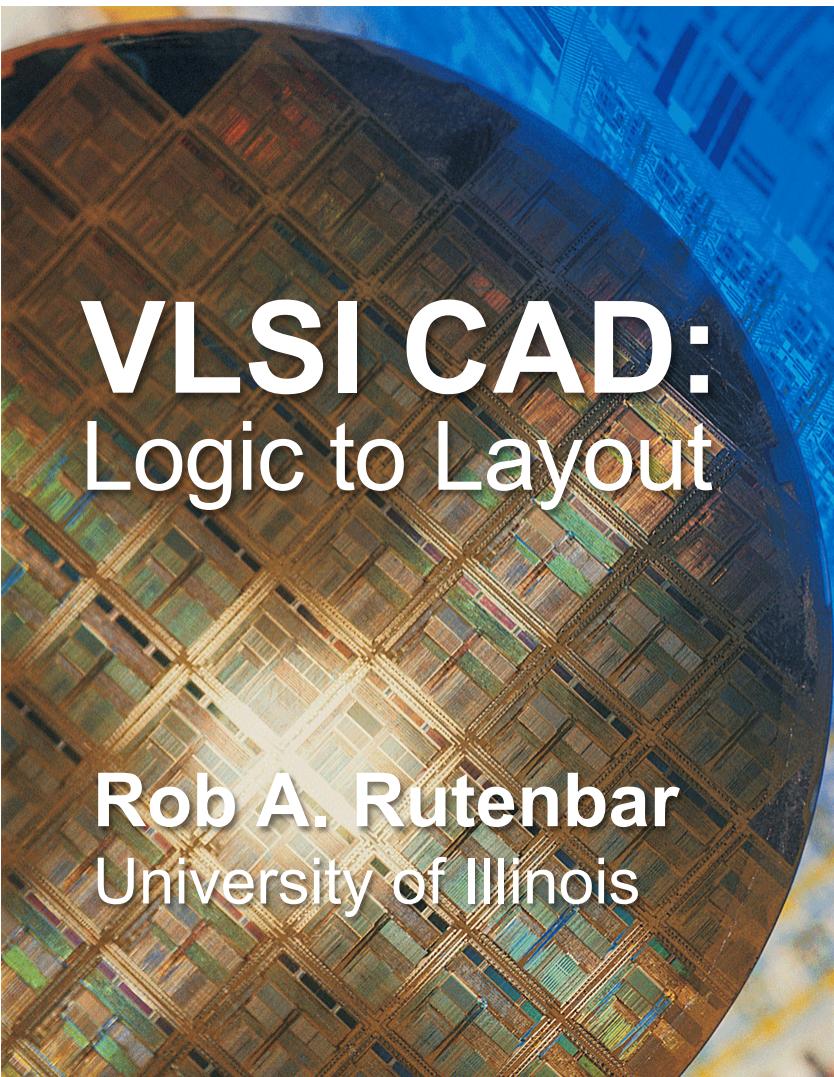
- Look what happened to **f**: Due to **network context**, it disappeared!



Summary

- **Don't Cares are implicit in the Boolean network model**
 - They arise from the graph structure of the multilevel Boolean network model itself
- **Implicit Don't Cares are powerful !**
 - They can greatly help simplify the 2-level SOP structure of any node
- **Implicit Don't Cares require computational work to go find**
 - For this example, we just “stared at the logic” to find the DC patterns
 - We need some **algorithms** to do this automatically!
 - This is what we need to study next!





VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

Lecture 8.3 Logic Synthesis: Multilevel Logic – Satisfiability Don't Cares



Chris Knott/Digital Vision/Getty Images

3 Types of Implicit DCs

- **Satisfiability don't cares: SDCs**
 - Belong to the wires inside the Boolean logic network
- **Controllability don't cares: CDCs**
 - Patterns that cannot happen at inputs to a network node
- **Observability don't cares: ODCs**
 - Patterns input to node that make network outputs insensitive to output of the node
 - Patterns that “mask” outputs
- **Let's see how we can compute all of these, automatically**



Background: Representing DC Patterns

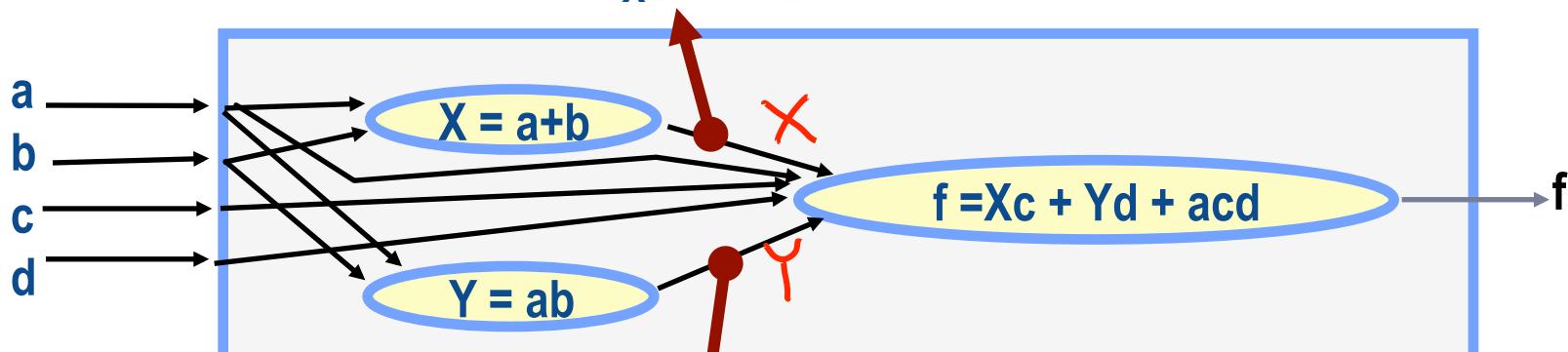
- **How will we represent don't care (DC) patterns at a node?**
 - **Answer:** As a **Boolean function** that makes a 1 when the pattern is **impossible**
 - This is often called a **Don't Care Cover**
 - So, each **SDC, CDC, ODC** is really just another Boolean function, in this strategy
- **Why do it like this?**
 - Because the **math** works (!)
 - But more importantly: we can use all the other **computational Boolean algebra** techniques we know (eg, BDDs), to **solve** for, and to **manipulate** the DC patterns
 - This turns out to be hugely important to making this practical



SDCs: They “Belong” to the Wires

- **One SDC for every internal wire in Boolean logic network**
 - The **SDC** represents impossible patterns of **inputs to, and output of**, each node
 - If the node function is **F**, with inputs **(a,b,c, ...)** write as: **SDC_F(F,a,b,c...)**

SDC_X(X,a,b) = 1 for impossible patterns of Xab



SDC_Y(Y,a,b) = 1 for impossible patterns of Yab



SDCs: How to Compute

- **Easy**

- Compute an **SDC** for each output wire from each internal Boolean node

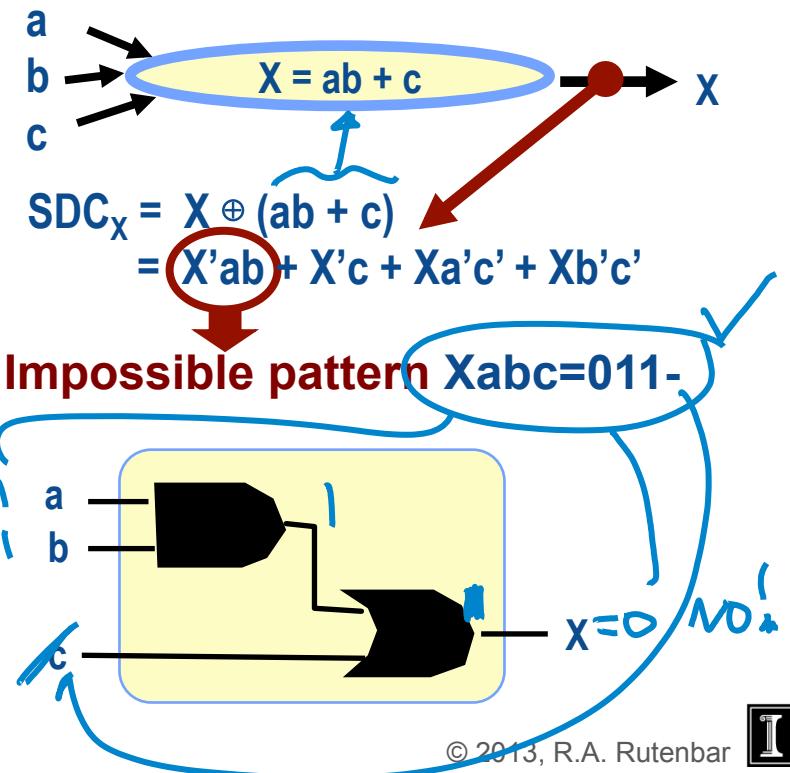
• You want an expression that's $\equiv 1$ when $X \neq$ (Boolean expression for X)

- This is just $[X \oplus (\text{expression for } X)]$

• Remember (**expression for X**) doesn't have X in it!!

- Yes: this is the **complement** of the gate consistency function from SAT

- **Example**

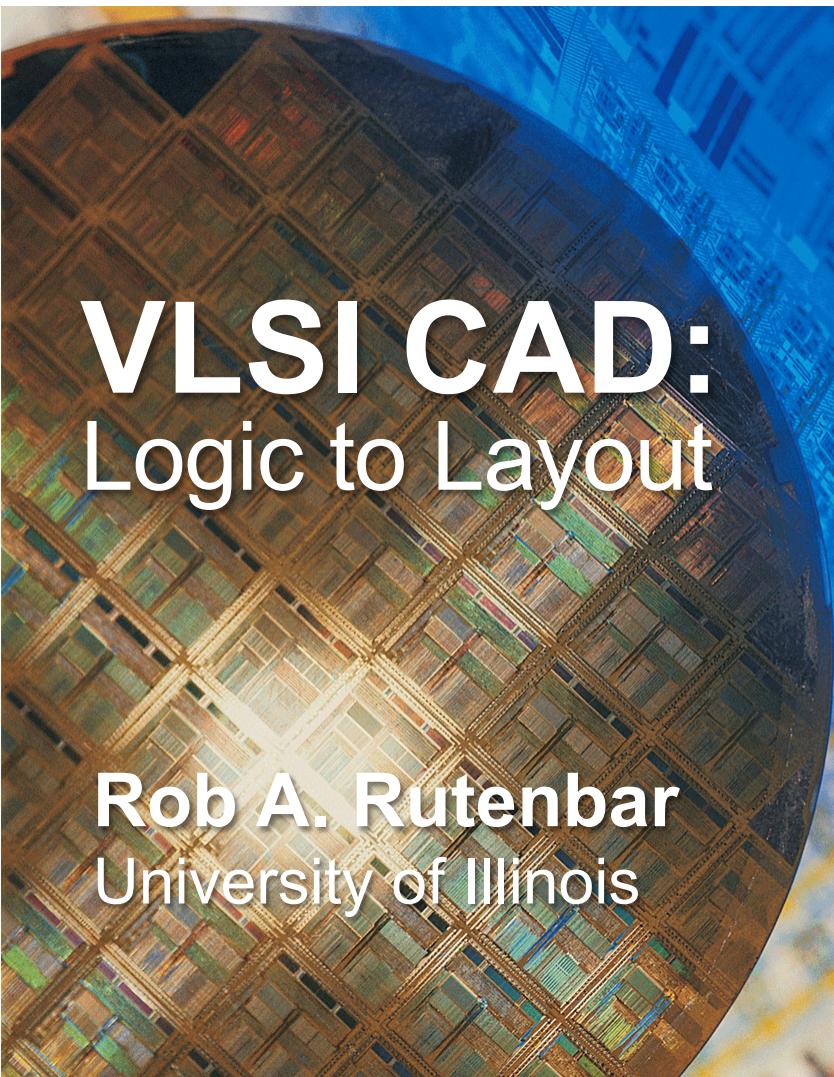


SDCs: Summary

- **SDCs associated with every internal wire in Boolean logic network**
 - SDCs explain impossible patterns of input to, and output of, each node
 - SDCs are easy to compute

- But SDCs alone are **not** the Don't Cares used to simplify nodes
 - We use SDCs to build CDCs, which give impossible patterns at input of nodes





VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

Lecture 8.4

Logic Synthesis: Multilevel Logic – Controllability Don't Cares



Chris Knott/Digital Vision/Getty Images

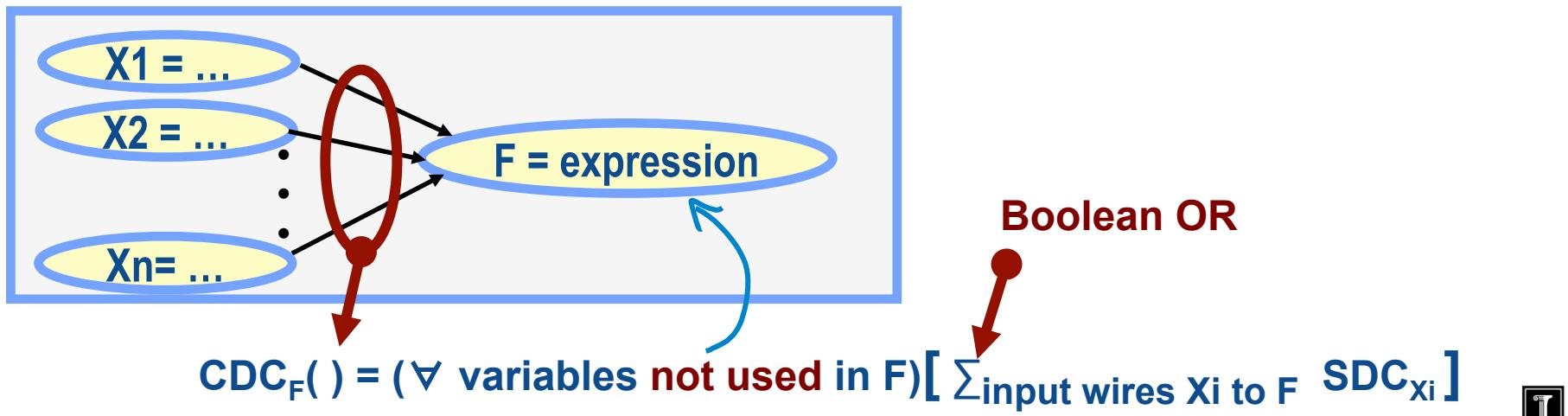
3 Types of Implicit DCs

- **Satisfiability don't cares: SDCs**
 - Belong to the wires inside the Boolean logic network
- **Controllability don't cares: CDCs**
 - Patterns that cannot happen at inputs to a network node
- **Observability don't cares: ODCs**
 - Patterns input to node that make network outputs insensitive to output of the node
 - Patterns that “mask” outputs
- **Let's see how we can compute all of these, automatically**

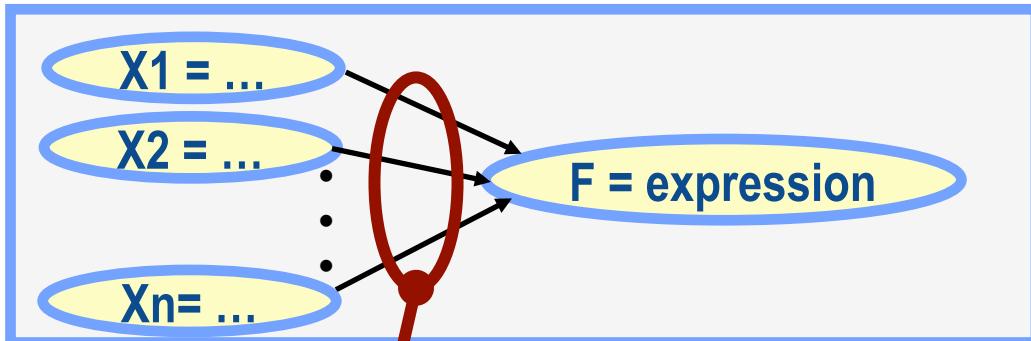


How to Get Impossible Patterns at Input to Node?

- Surprisingly simple computational recipe for CDC_F :
 - Get all the **SDCs** on the wires input to this node in Boolean logic network 
 - **OR** together all these **SDCs**
 - **Universally Quantify** away all variables that are **NOT** used inside this node
 - **Result:** Boolean function (CDC_F below) == 1 for impossible patterns at input to node!



CDCs: Why Does This Work?



$$\text{CDC}_F() = (\forall \text{ variables not used in } F) [\sum_{\text{input wires } X_i \text{ to } F} \text{SDC}_{X_i}]$$

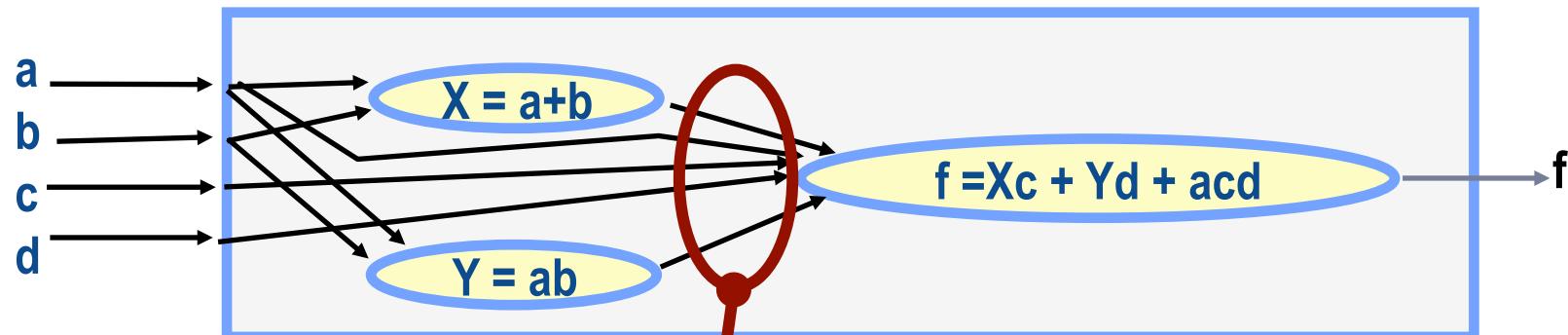
- **Roughly speaking...**
 - SDC_{X_i} 's explain all the impossible patterns involving X_i wires input to the F node
 - **OR** operation is just the "union" of all these impossible patterns involving X_i 's
 - **Universal Quantify** removes variables **not** used by F , and does so in the right way: want patterns that are impossible **FOR ALL** values of these removed variables

+



Real Example: SDCs \rightarrow CDC_f

- Lets apply this computational scheme to the f node below

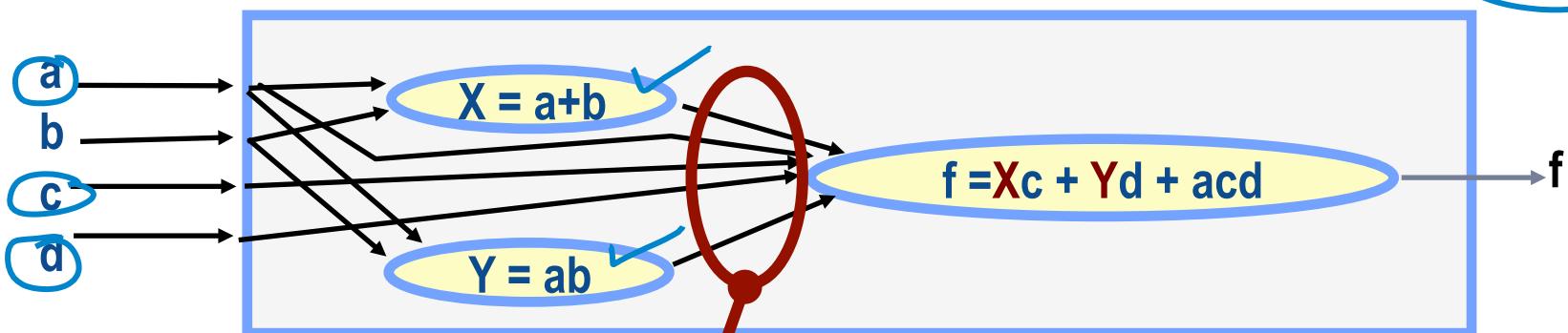


$$CDC_f = \underbrace{\left(\forall \text{ variables not input to } f \right)}_{\text{This is variable } b} \underbrace{\left(\sum_{\text{inputs } i} \text{SDC}_i \right)}_{\text{Input variables to } f \text{ are } X, Y, a, c, d}$$



SDCs \rightarrow CDCs

- What about SDCs on primary inputs (no network node for them)?
 - We can ignore SDCs on a, c, d inputs to f since they are primary inputs.
 - Why? $SDC_a = a \oplus (\text{expression for } a) = a \oplus a = 0$. No impact on OR. Ignore.



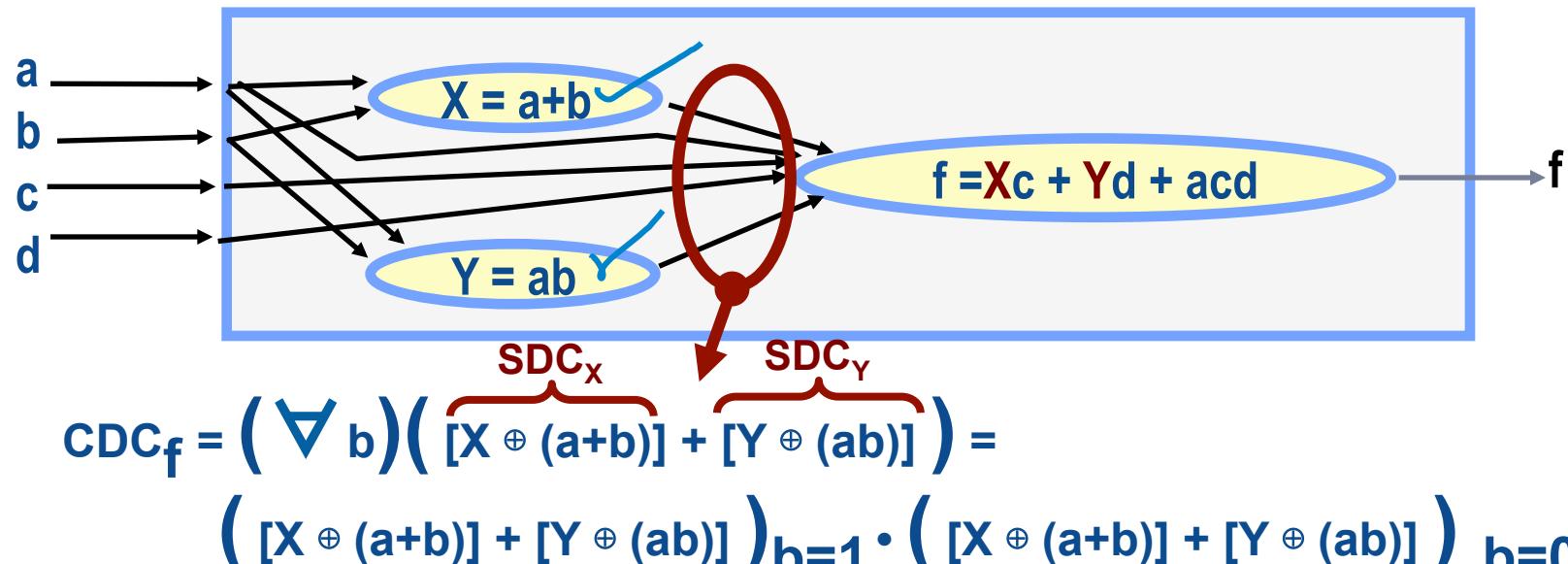
$$CDC_f = \left(\forall \text{ variables not input to } f \right) \left(\sum_{\text{inputs } i} SDC_i \right)$$

Only X, Y



SDCs → CDCs

- Do the (computational) Boolean algebra...

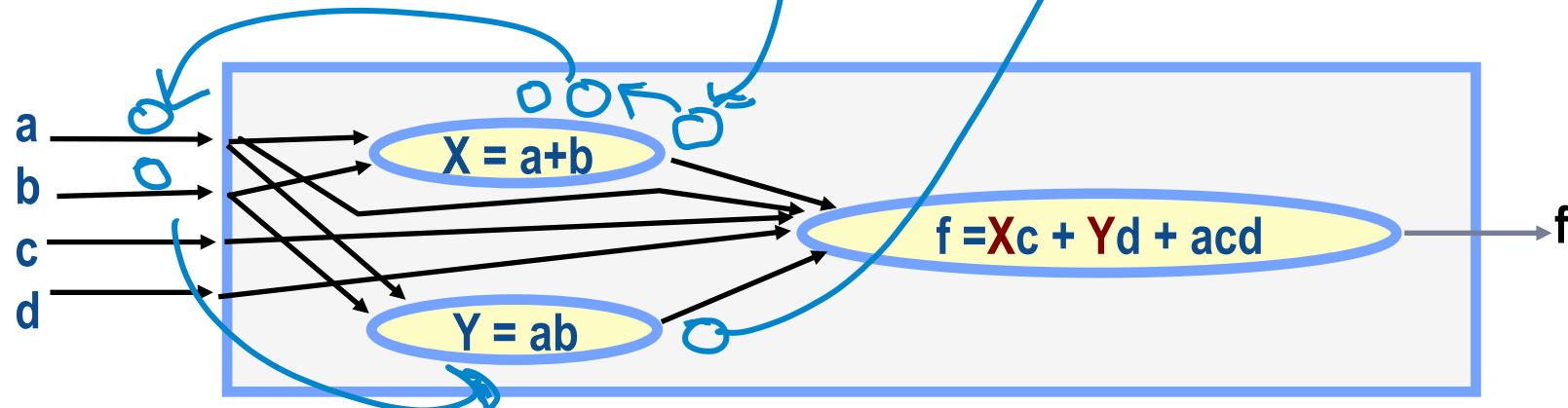


Result = $X'a + X'Y + Y'a$ are impossible patterns for f



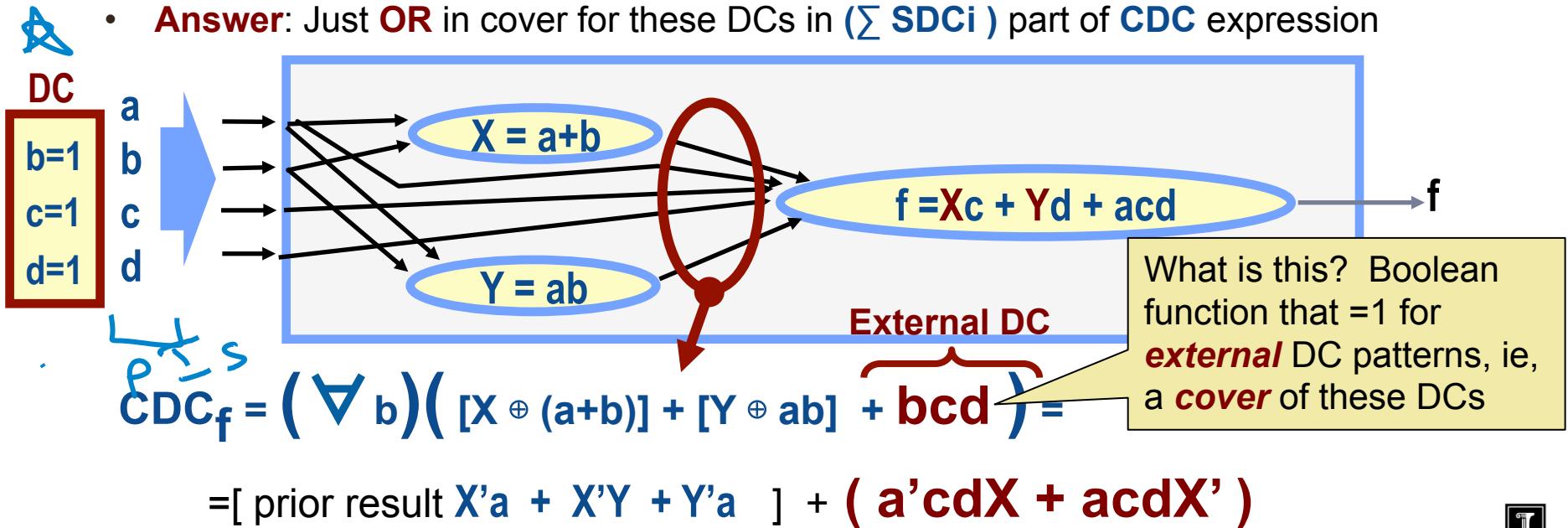
CDC_f – Does It Make Sense?

- $CDC_f(X, Y, a, c, d) = X'a + \textcircled{X'Y} + Y'a \rightarrow XY = 01$ is impossible for f
 - This makes sense. Why?
 - $X = 0 \rightarrow a=0$ and $b=0 \rightarrow Y = 0$ so impossible to have $X=0 \& Y=1$!



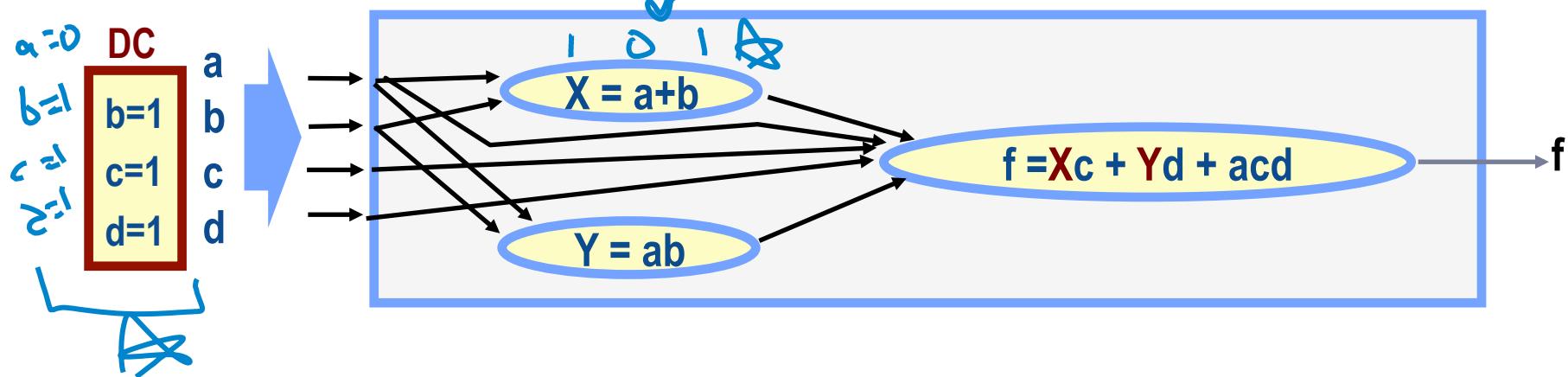
Yet More CDCs: *External* CDCs

- Next: what if there are actually DCs for network inputs, *external* patterns for $abcd$ where we just *do not care* what f does?
 - Example: suppose $b=1 c=1 d=1$ cannot happen. How to compute CDC_f now?
 - Answer: Just **OR** in cover for these DCs in $(\sum \text{SDCi})$ part of **CDC** expression



External CDCs: Does It Make Sense?

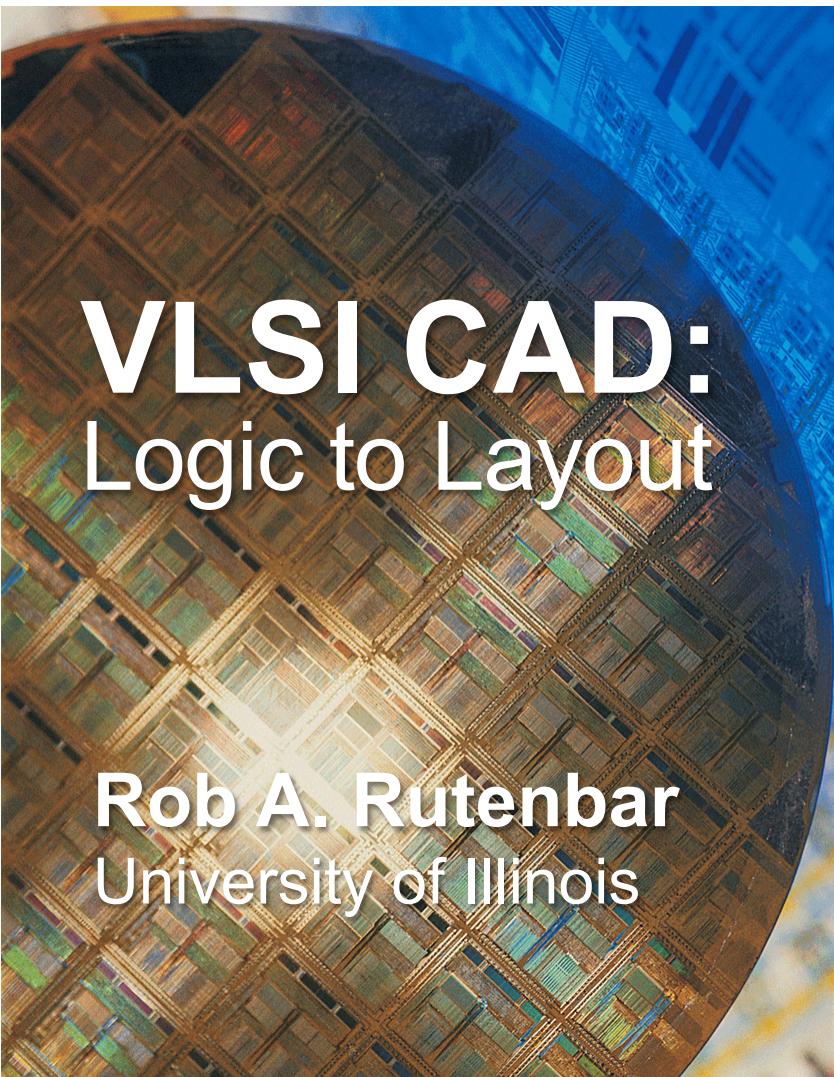
- New $CDC_f = X'a + X'Y + Y'a + a'cdX + acdX'$ impossible for f
- $a'cdX=1 \rightarrow a=0 \&& X=1 \rightarrow b \text{ must be } 1 \rightarrow b=1 \&& c=1 \&& d=1 \rightarrow \text{Don't Care!}$



CDCs: Summary

- **CDCs give impossible patterns at input to node F – use as DCs**
 - Impossible because of the network structure of the nodes **feeding** node F
 - **CDCs** ~~are~~ can be computed mechanically from **SDCs** on wires input to F
 - **Internal local CDCs:** computed just from **SDCs** on wires into F
 - **External global CDCs:** DC patterns at network input, can be included too
- **But CDCs still *not all* the Don't Cares available to simplify nodes**
 - **CDC_F** derived from the structure of nodes “**in front of**” node F
 - We need to look at DCs that derive from nodes “**in back of**” node F
 - These are nodes between the output of F and primary outputs of overall network





VLSI CAD: Logic to Layout

Rob A. Rutenbar
University of Illinois

Lecture 8.5

Logic Synthesis: Multilevel Logic – Observability Don't Cares



Chris Knott/Digital Vision/Getty Images

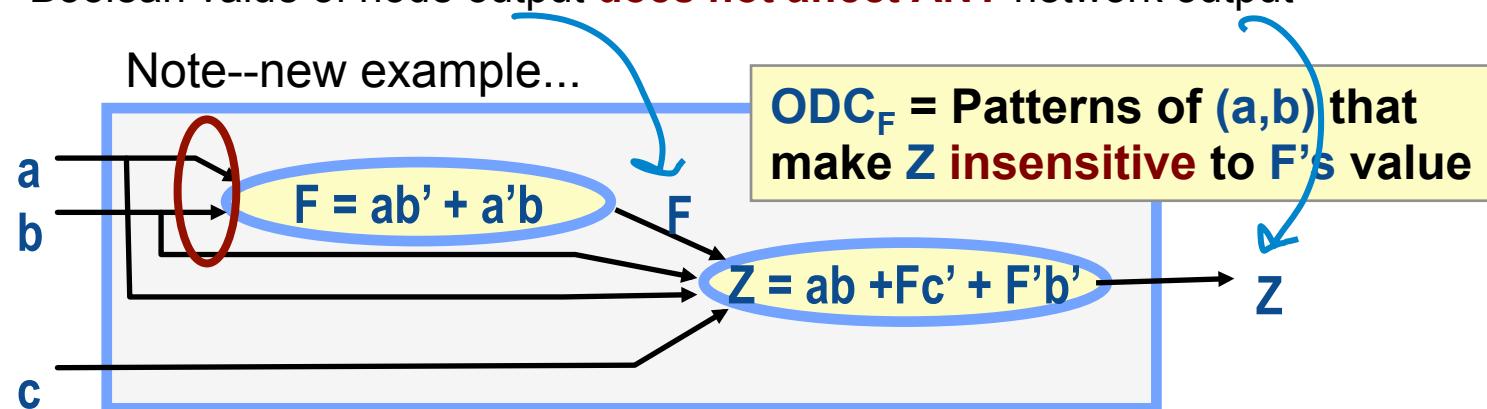
3 Types of Implicit DCs

- **Satisfiability don't cares: SDCs**
 - Belong to the wires inside the Boolean logic network
- **Controllability don't cares: CDCs**
 - Patterns that cannot happen at inputs to a network node
- **Observability don't cares: ODCs**
 - Patterns input to node that make network outputs insensitive to output of the node
 - Patterns that “mask” outputs
- **Let's see how we can compute all of these, automatically**



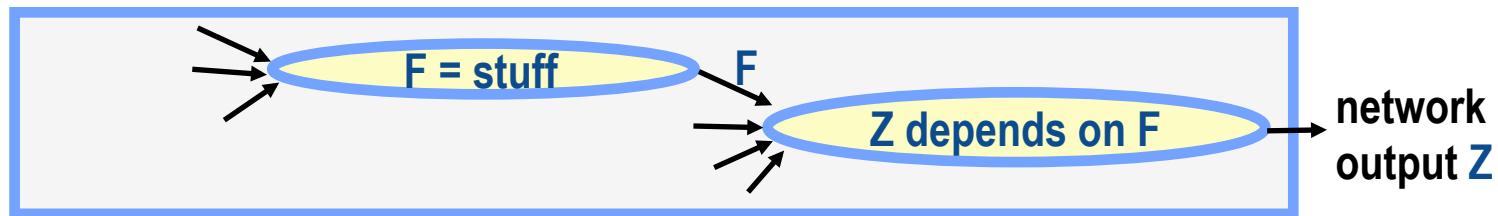
Final DCs: *Observability* Don't Cares

- ODCs = patterns that **mask** a node's output at network output
 - So, these are **not impossible** patterns – these patterns **can occur** at node input
 - These patterns make this node's output **not observable** at network output
- “**Not observable**” for an input pattern means...
 - Boolean value of node output **does not affect ANY** network output

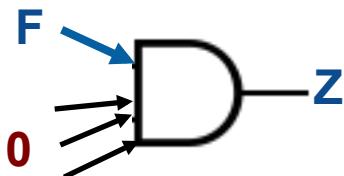


Network Output *Insensitive* to F

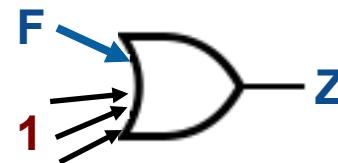
- When is network output **Z** *insensitive* to internal variable **F**?



- Means **Z** *independent* of value of **F**, given **other inputs** to **Z**



Z insensitive to **F**
if any other **input = 0**



Z insensitive to **F**
if any other **input = 1**



How to **compute**
the general case...?



We've (Almost) Seen This Before!

- What makes output **F** sensitive to input **X**?
 - Any pattern that makes $\partial F / \partial X = 1$ makes **X observable** at output **F**

Slide 21
Lecture 2.6

Interpreting the Boolean Difference

$\frac{\partial F}{\partial x} = 1$

a, b, c, ..., w

A BIG BLOB OF LOGIC

$\Delta F(a, b, c, \dots, w, x)$

When $\partial F / \partial x (a, b, c, \dots, w) = 1$, it means that ...

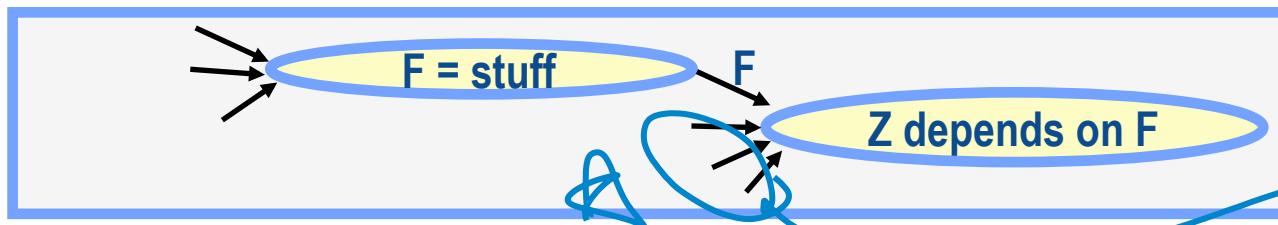
If you apply a pattern of other inputs (not **X**) that makes $\partial F / \partial x = 1$, Any change in **X** will force a change in output **F()**

Slide 21

© 2013, R.A. Rutenbar

Z Insensitive to F When $(\partial Z / \partial F) = 1$

- When is network output Z *insensitive* to internal variable F ?



Answer: When inputs to Z make cofactors $Z(F=0) = Z(F=1)$ (ie, $Z_F = Z_{F'}$)

Makes **sense**: if cofactors with respect to F **same**, Z does **not** depend on F !

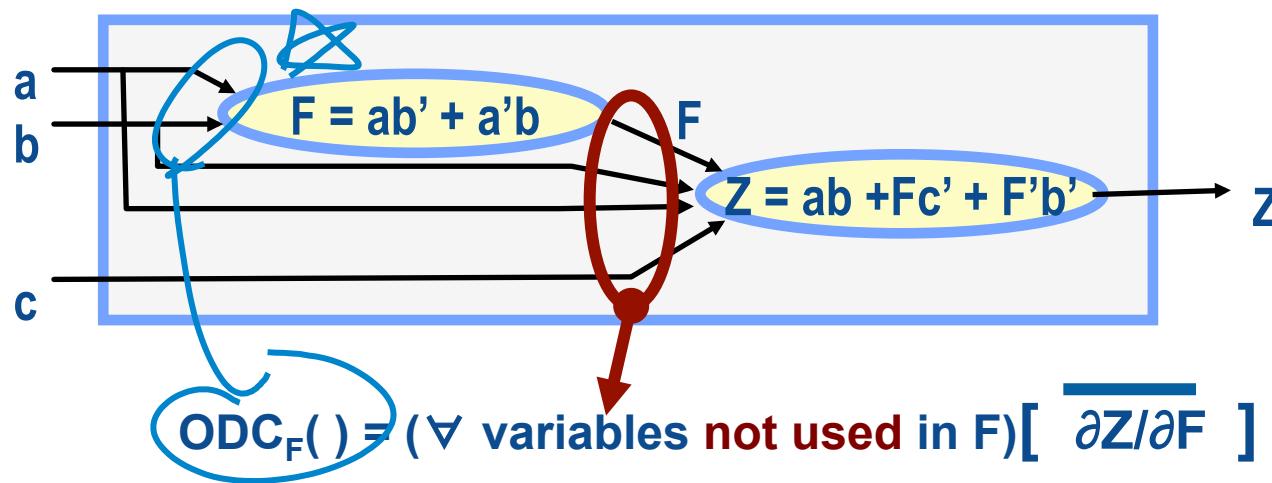
How to **find** when cofactors are the same? **Solve** for $Z(F=0) \oplus Z(F=1) = 1$

Note: $Z(F=0) \oplus Z(F=1) = 1$ same as $\underbrace{[Z(F=0) \oplus Z(F=1)]}_{\partial Z / \partial F !} = 1 \rightarrow \boxed{(\partial Z / \partial F) = 1}$



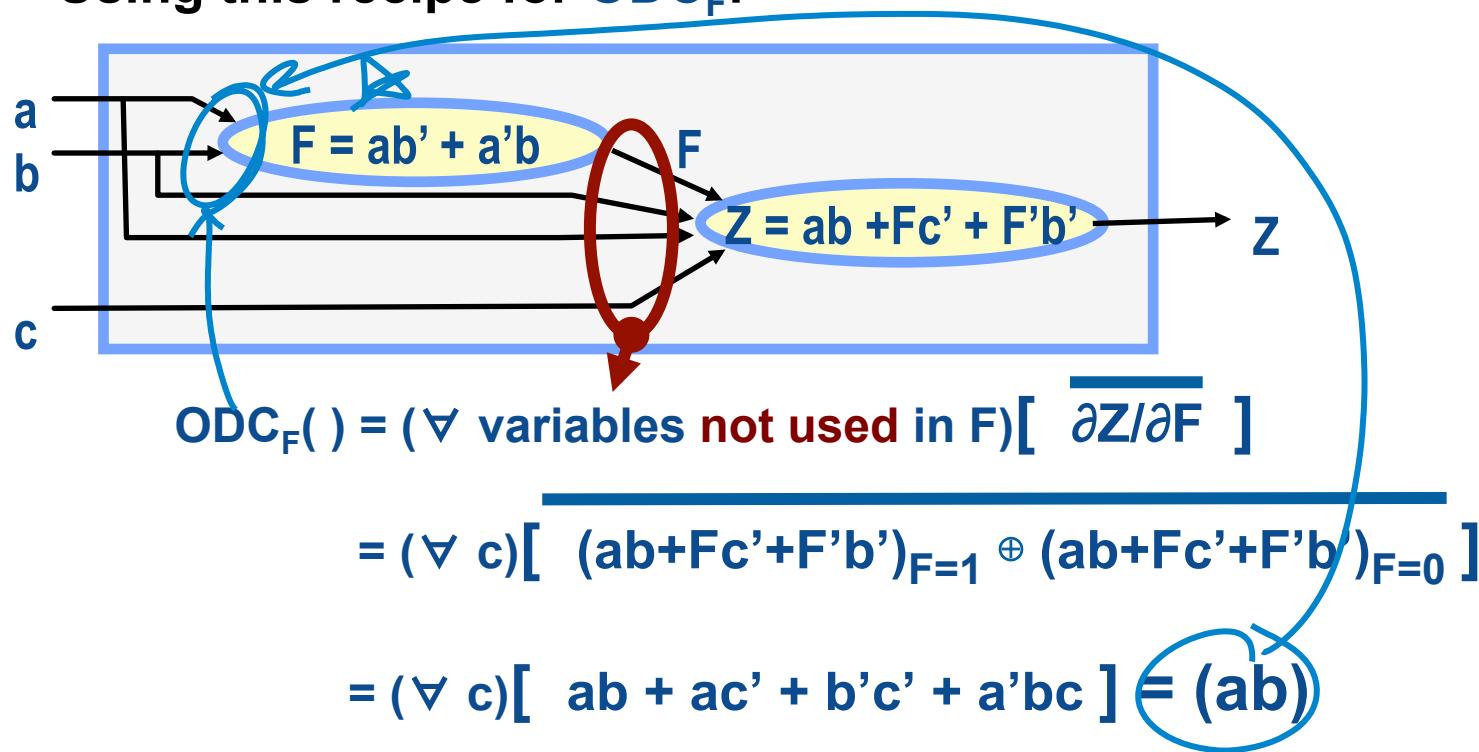
How to Get ODC Patterns that Mask Node Output?

- Again, a nice computational recipe for ODC_F :
 - Compute $\overline{\partial Z / \partial F}$. Any patterns that make $\overline{\partial Z / \partial F} == 1$ mask output F for Z .
 - Universally Quantify away all variables that are NOT inputs to the F node itself
 - Result: Boolean function (ODC_F below) == 1 for “masking” patterns at input to node!



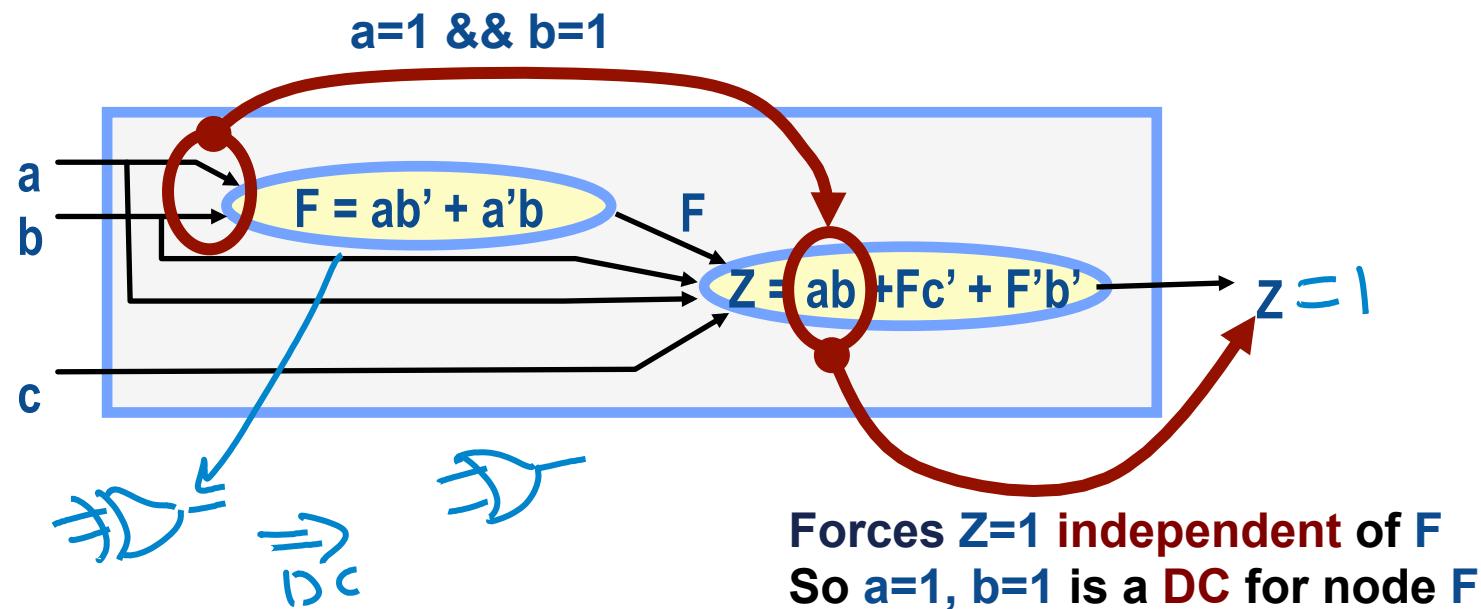
How to Get ODC Patterns that Mask Node Output?

- Using this recipe for ODC_F :



Check: Does this ODC Make Sense?

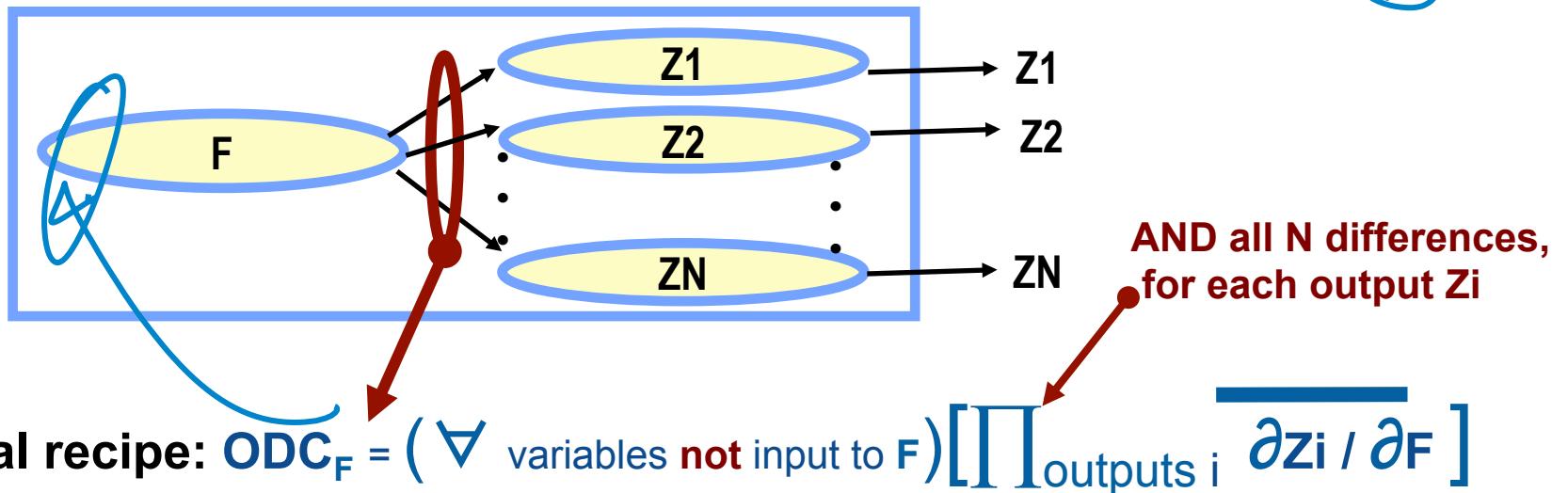
- Yes: $ODC_F(a,b) = ab \rightarrow a=1, b=1$ masks **F** from affecting **Z** output



ODCS: More General Case

- Question: what if **many** network outputs?

- Answer: Only patterns that are unobservable at **ALL** outputs can be **ODCs**



= patterns that make **F** unobservable at **ALL** **Zi** outputs

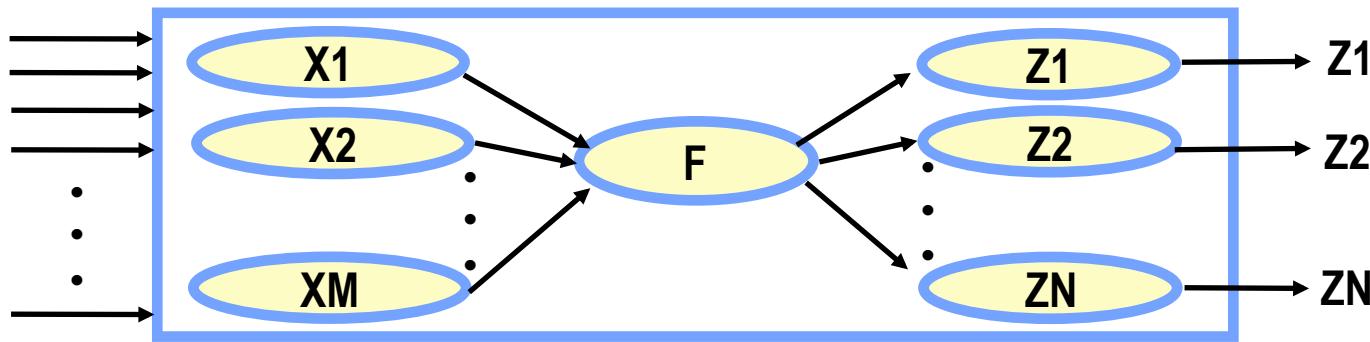


ODCs: Summary

- **ODCs give patterns input to node F that mask F at network output**
 - **Not** impossible patterns – they **can** occur.
 - Don't cares because network output “doesn't care” what **F** is, for these patterns
 - **ODCs** are can be computed mechanically from $\partial Z_i / \partial F$'s on all outputs connected to **F**
- **CDCs + ODCs give the “full” don't care set used to simplify F**
 - With these patterns, you can call something like ESPRESSO to simplify **F**



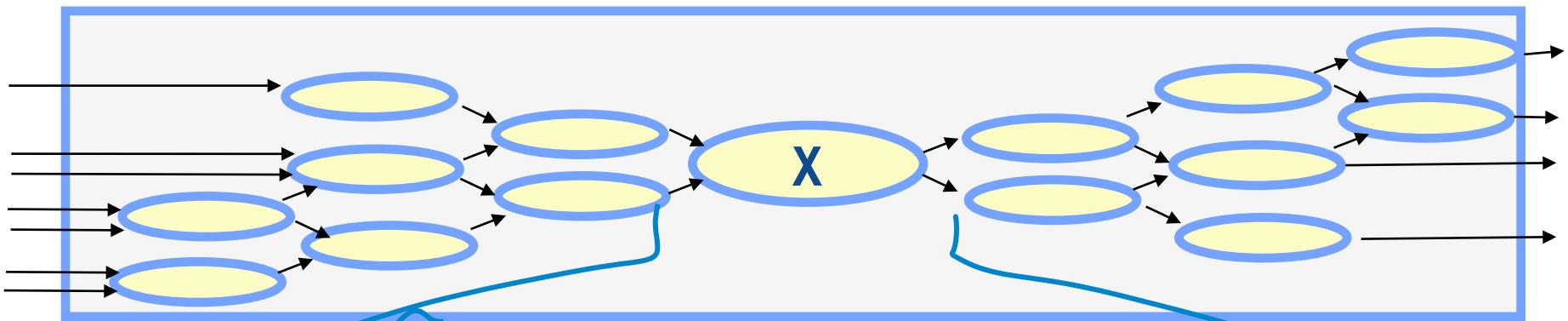
Multi-Level Don't Cares: Are We Done?



- Yes – if your networks look **just like this**
 - More precisely, if you only want to get **CDCs** from nodes immediately “before” you
 - And if you only want to get **ODCs** for **one layer of nodes between** you and output



Don't Cares, In General



- But, this is what real multi-level logic can look like (!)
 - CDCs are a function of **all nodes** “in front” of **X**
 - ODCs a function of **all nodes** between **X** and any output
 - In general, we can **never get all** the DCs for node **X** in a big network
 - Representing all this stuff can be explosively large, **even** with BDDs



Summary: Getting Network DCs

- **How we really do it:** generally **do not get all the DCs**
 - Lots of tricks that trade off effort (time, memory) vs. quality (how many **DCs**)
 - **Example:** Can just extract “local **CDCs**”, which requires looking at outputs of immediate antecedent vertices and computing from the **SDC** patterns, which is easy
 - There are also incremental, node-by-node algorithms that walk the network to compute more of the **CDC** and **ODC** set for **X**, but these are more **complex**
- **For us, knowing these “limited” DC recipes is sufficient**
 - You now understand why **DCs** are implicit, why you must find them, and the big differences between **SDCs**, **CDCs**, and **ODCs**. This is good!

