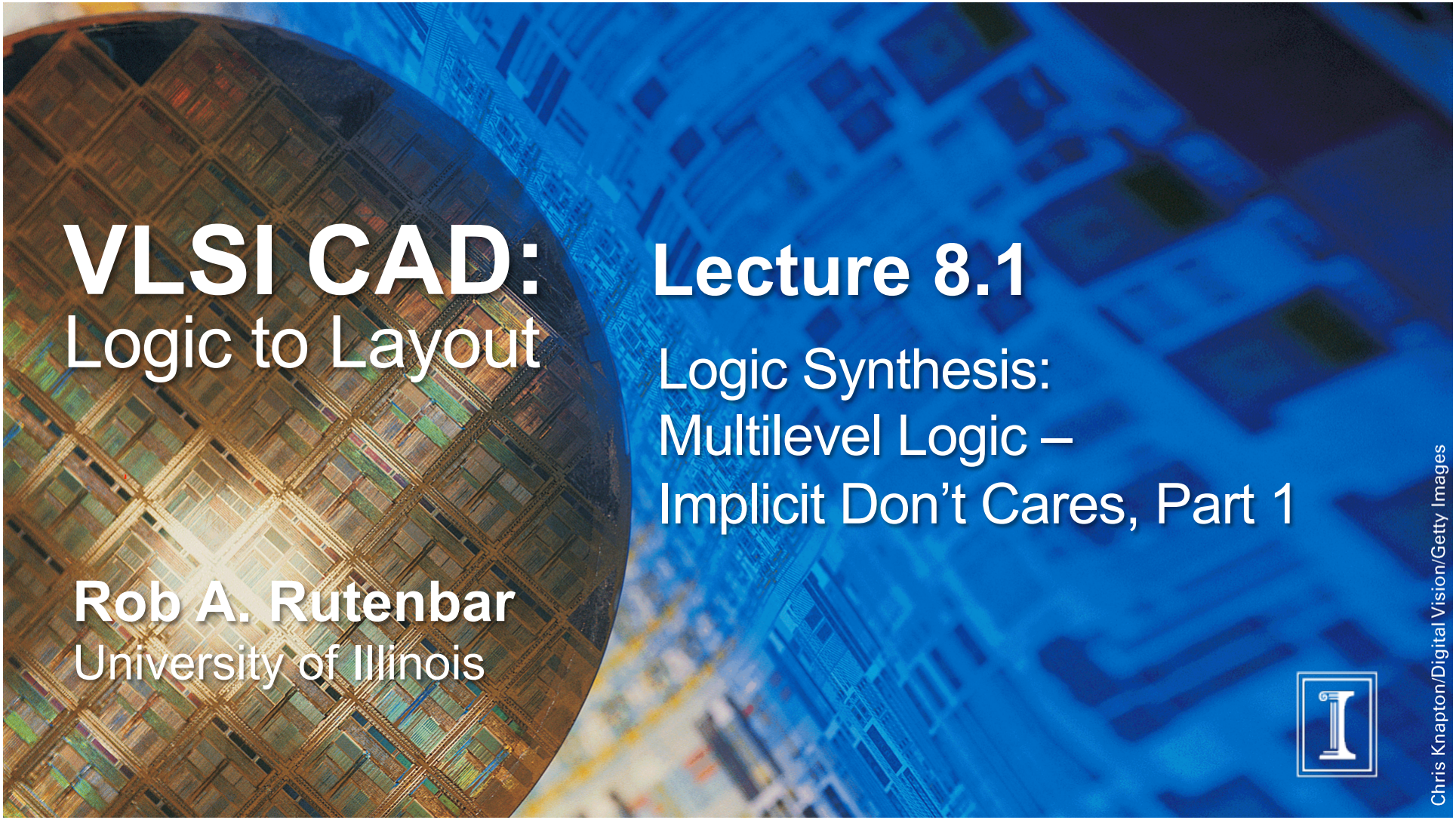


VLSI CAD: Logic to Layout

Lecture 8.1

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

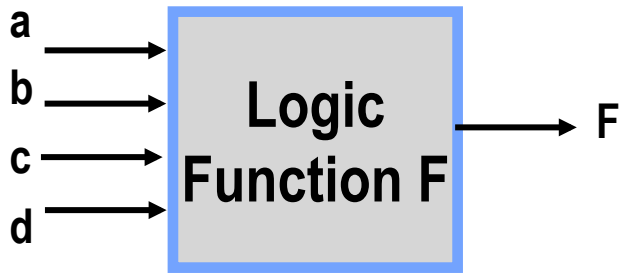
Rob A. Rutenbar
University of Illinois

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



cd \ ab	00	01	11	10
00			d	1
01			d	1
11		1	d	d
10			d	d

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**

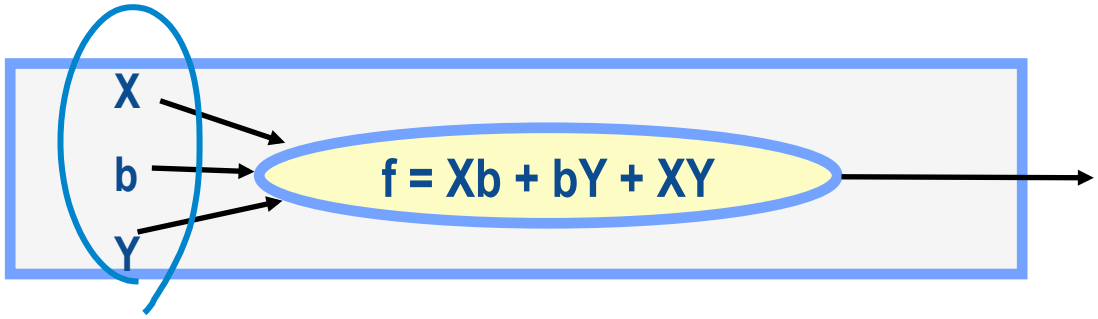
Don’t Cares (DCs): Multi-level

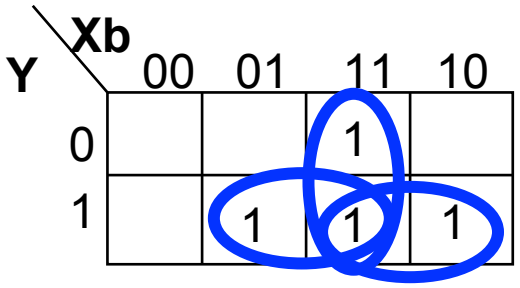
- **What’s different in multi-level?**
  - **DCs** arise **implicitly**, as a result of the Boolean logic network structure
  - We must go **find** these implicit don’t cares – we must search for them explicitly

- **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

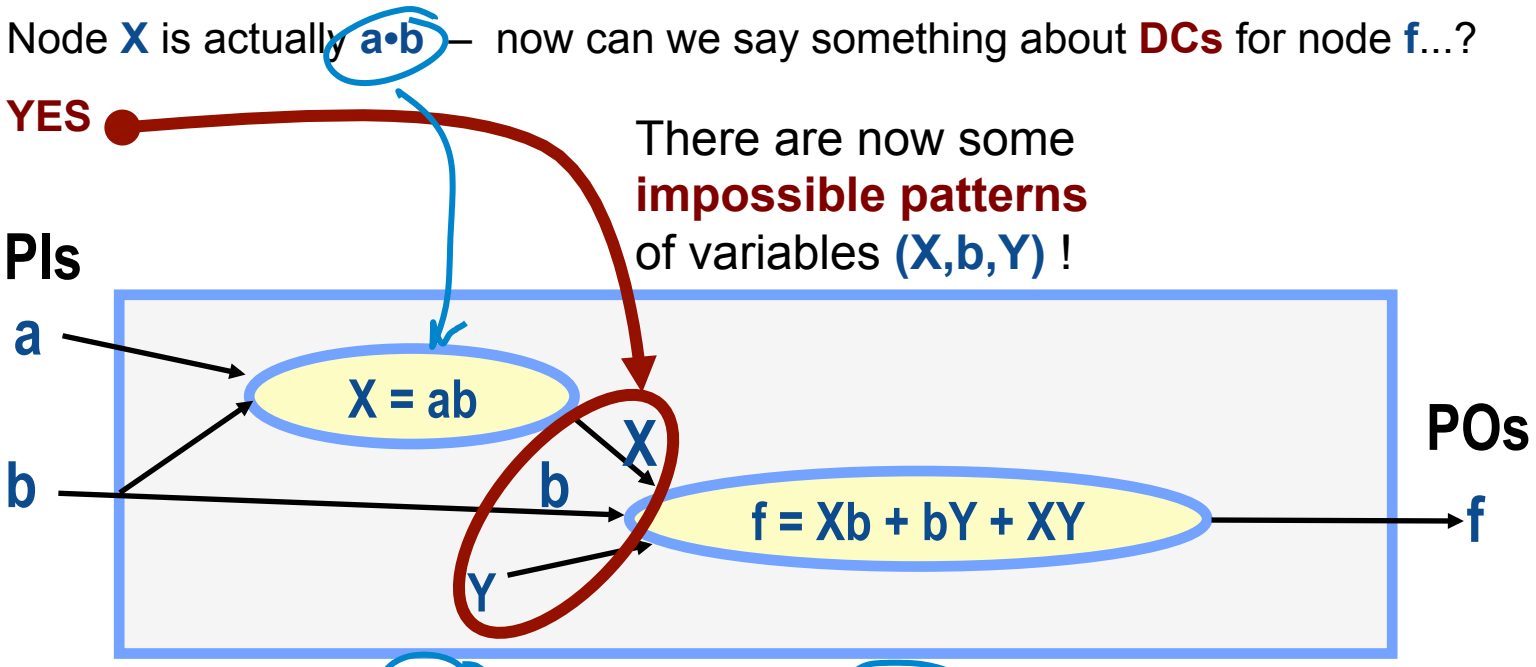
- **Suppose we have a Boolean network: lets look at node "f"**
- **Can we say anything about don't cares for node f?**
  - **No:** we don't know any "context" for surrounding parts of network
  - As far as we can tell, all patterns of inputs **(X,b,Y)** are **possible**





Multi-level DCs: Informal Tour

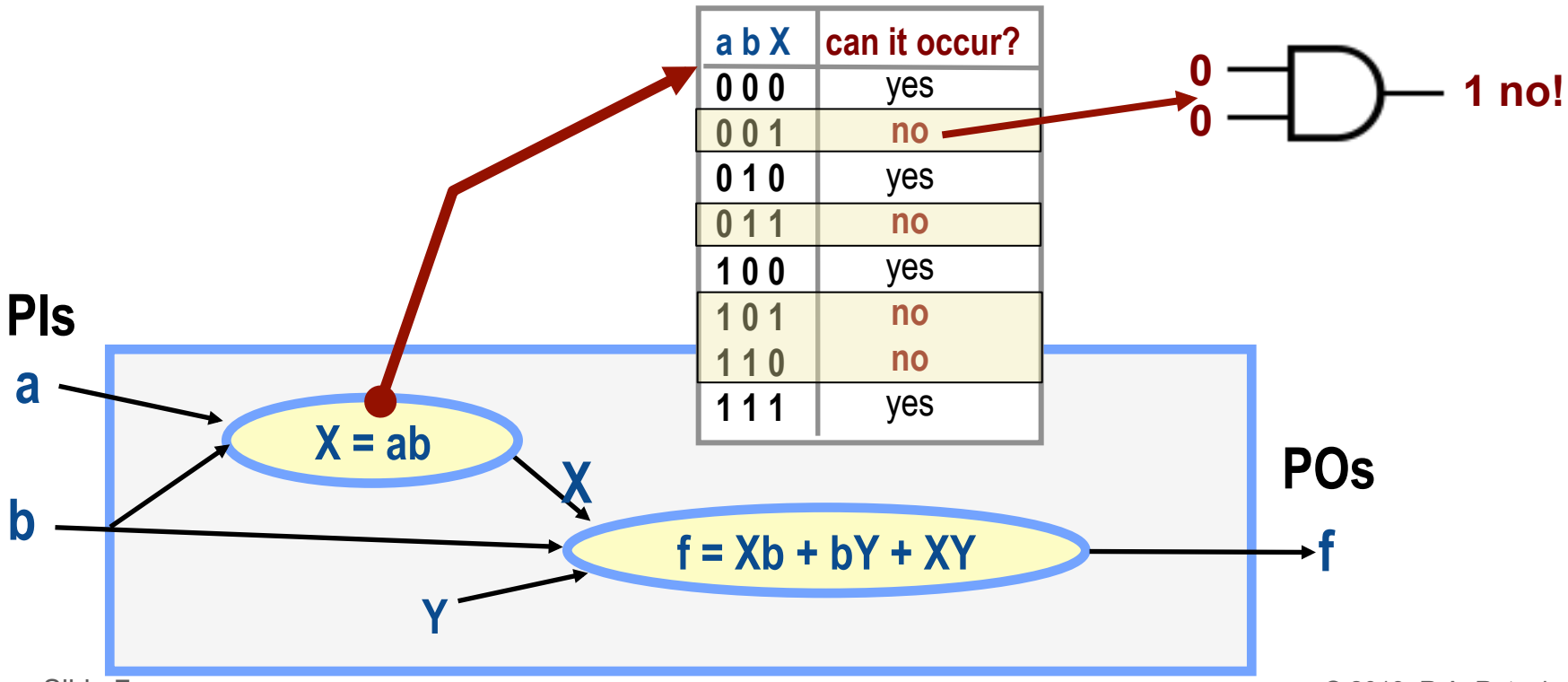
- **OK, suppose we know this about input X to f**
  - Node **X** is actually **a•b** – now can we say something about **DCs** for node **f**...?
  - **YES**



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

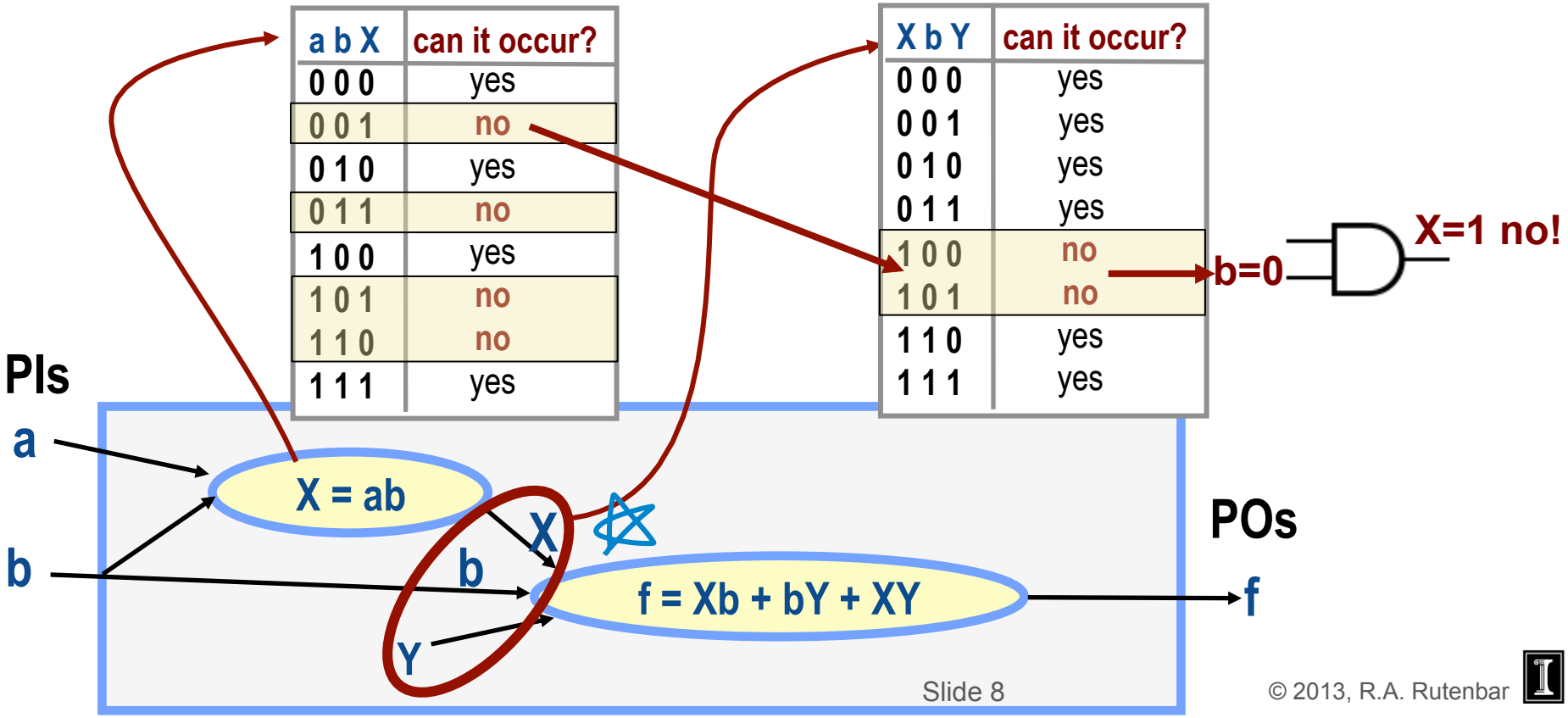
Multi-level DCs: Informal Tour

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



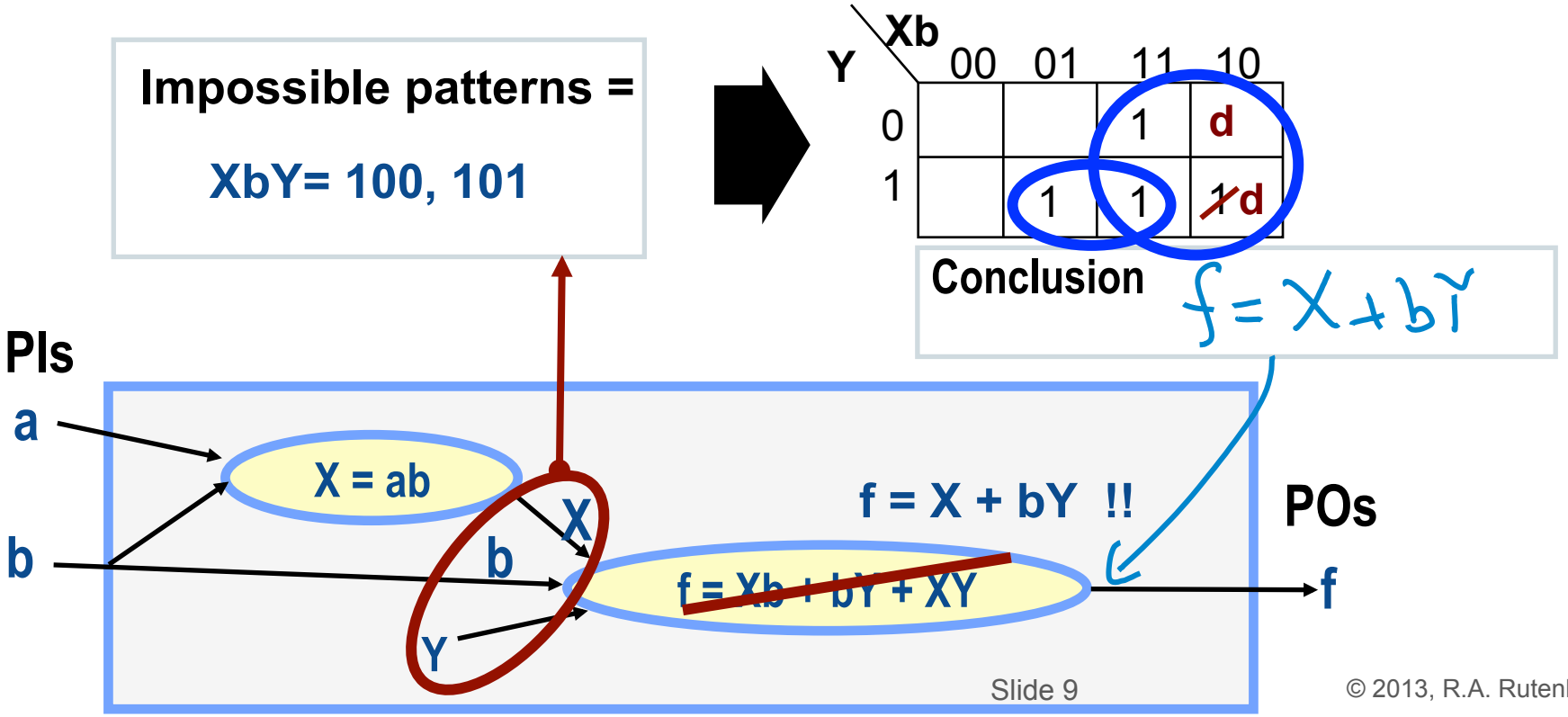
Multi-level DCs: Informal Tour

- **Impossible a b X patterns → Impossible X b Y patterns?**



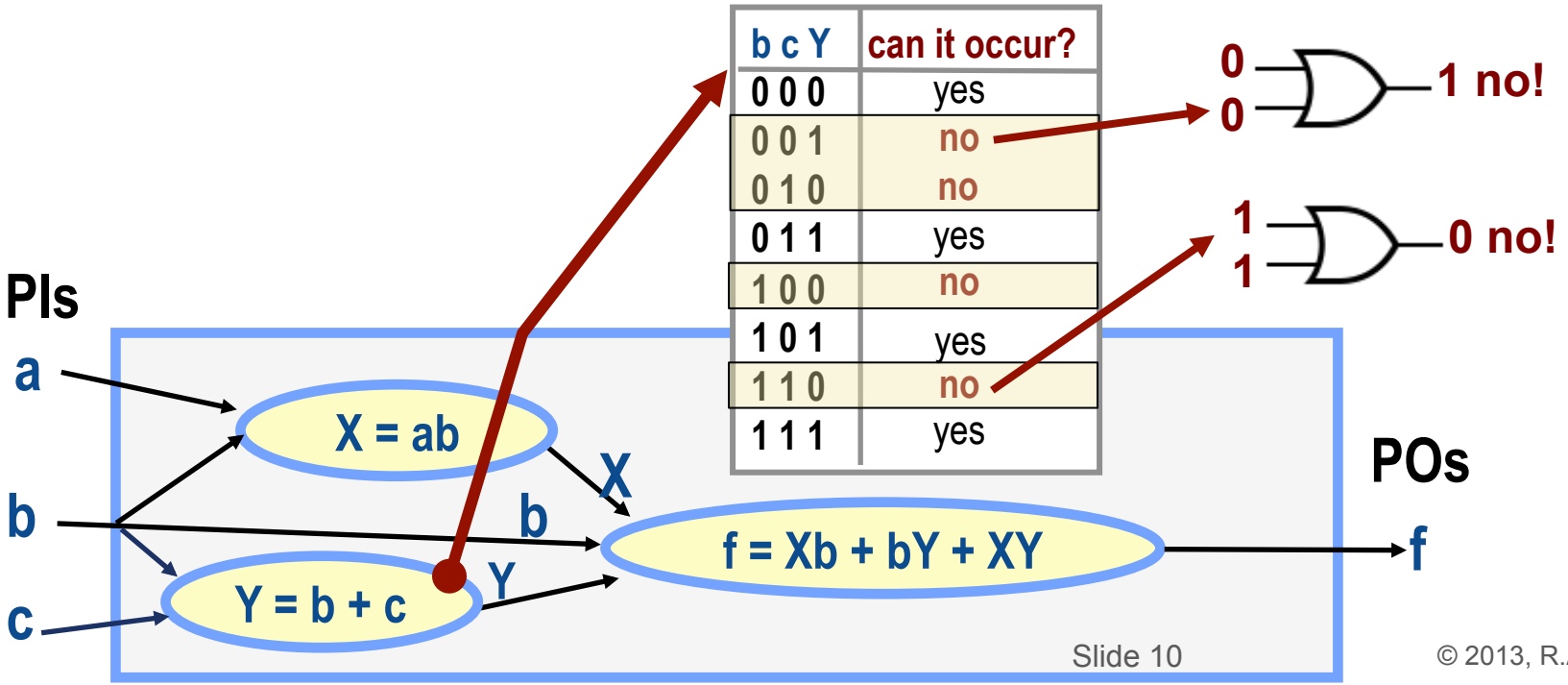
Impossible X b Y Patterns Give DCs for Node f

- 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** ?



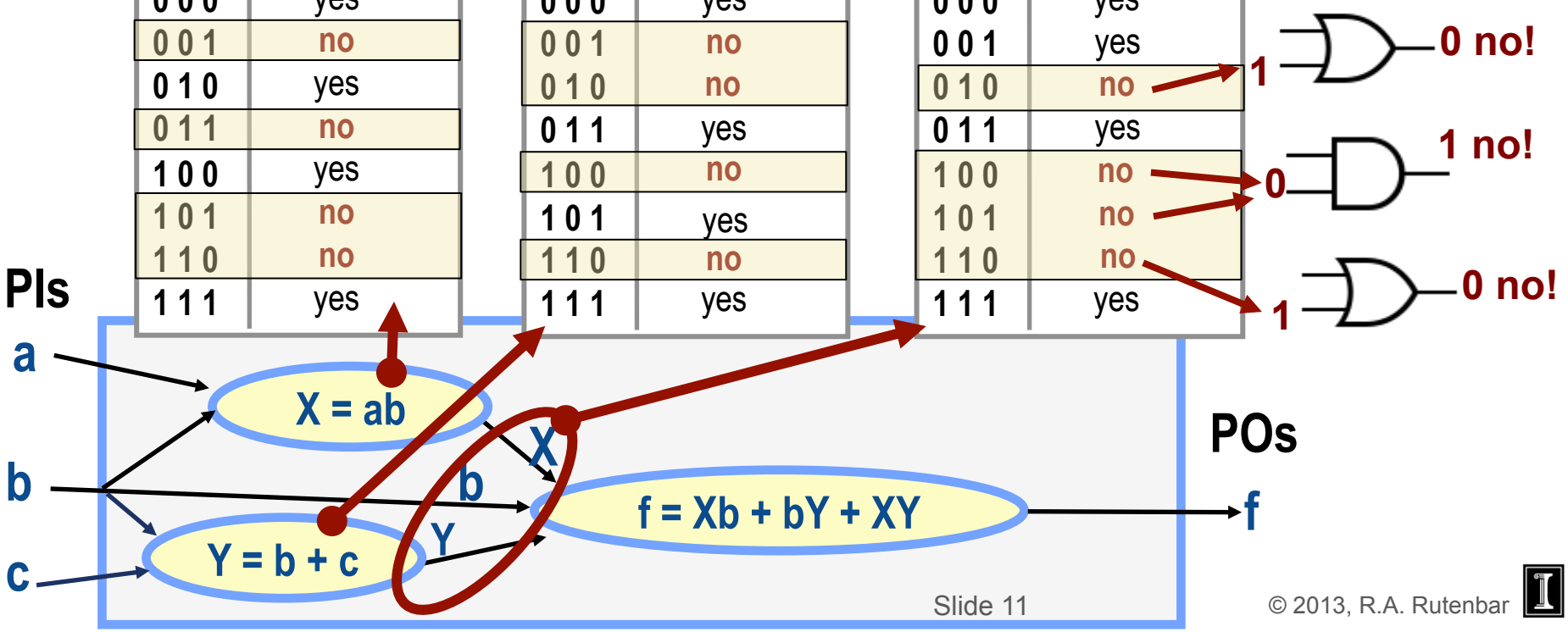
From Impossible Patterns of X,Y → f

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

a b X	can it occur?
0 0 0	yes
0 0 1	no
0 1 0	yes
0 1 1	no
1 0 0	yes
1 0 1	no
1 1 0	no
1 1 1	yes

b c Y	can it occur?
0 0 0	yes
0 0 1	no
0 1 0	no
0 1 1	yes
1 0 0	no
1 0 1	yes
1 1 0	no
1 1 1	yes

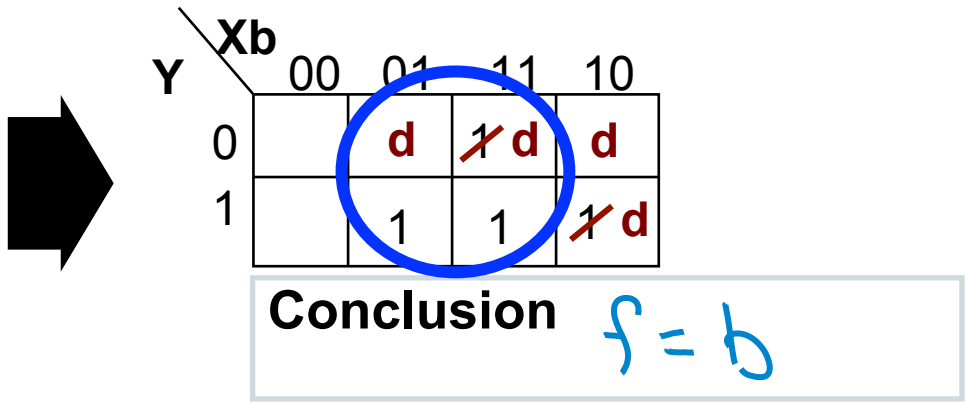
X b Y	can it occur?
0 0 0	yes
0 0 1	yes
0 1 0	no
0 1 1	yes
1 0 0	no
1 0 1	no
1 1 0	no
1 1 1	yes

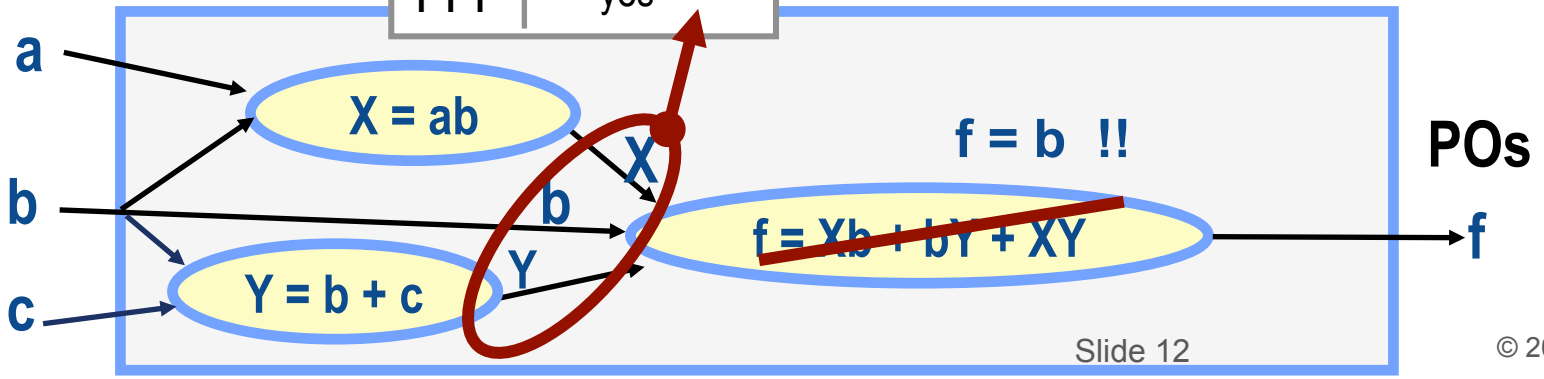


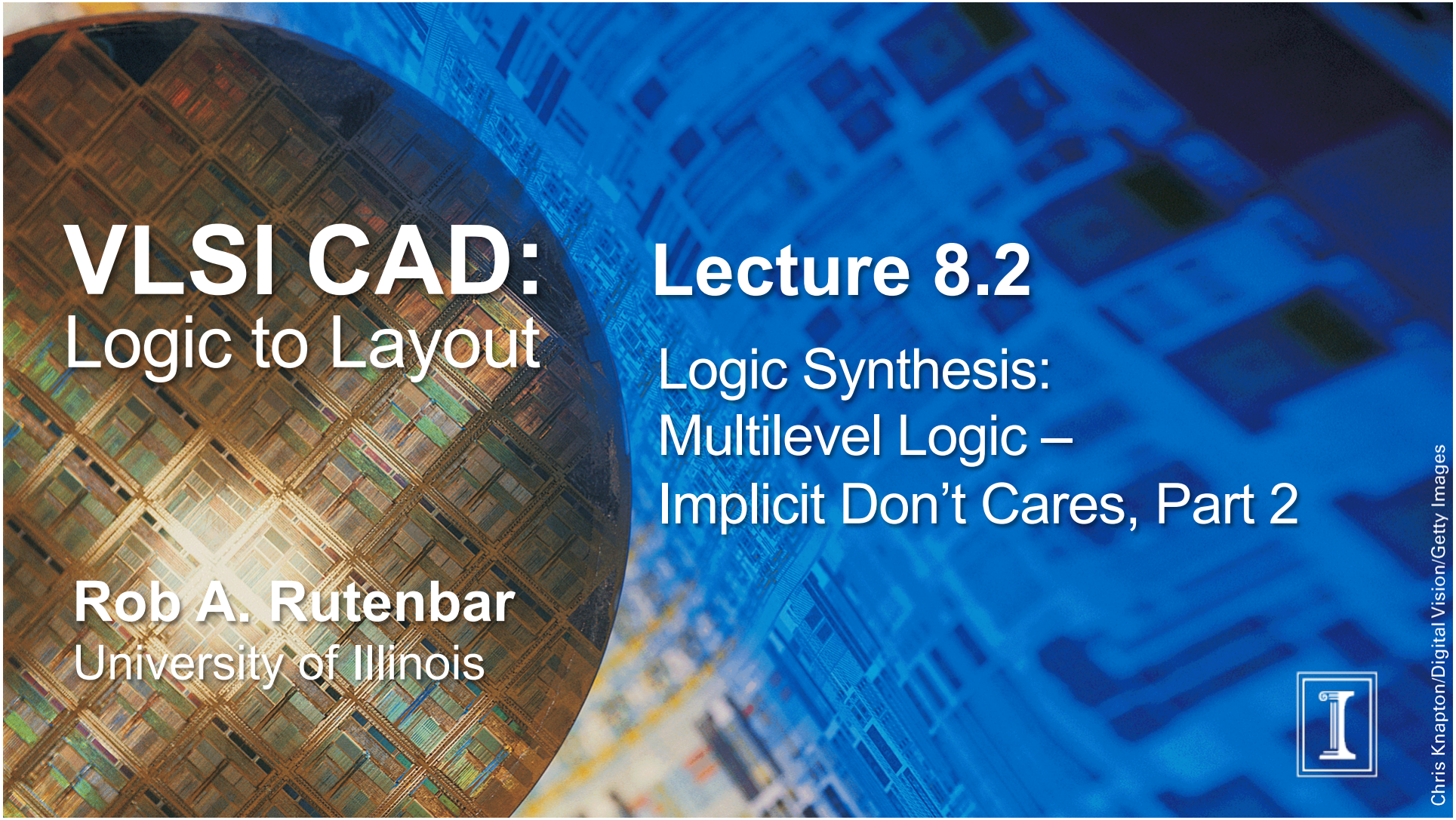
Informal Multi-level DC Tour

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

X b Y	can it occur?
0 0 0	yes
0 0 1	yes
0 1 0	no
0 1 1	yes
1 0 0	no
1 0 1	no
1 1 0	no
1 1 1	yes







VLSI CAD: Logic to Layout

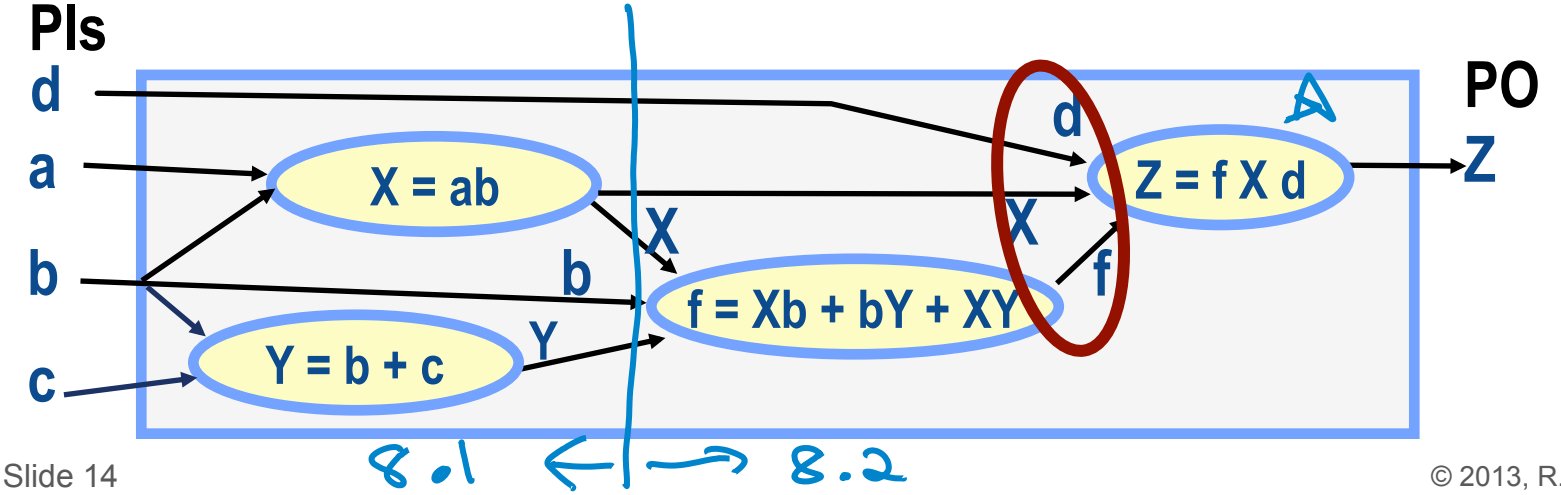
Lecture 8.2

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

Rob A. Rutenbar
University of Illinois

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 X d** at node Z...

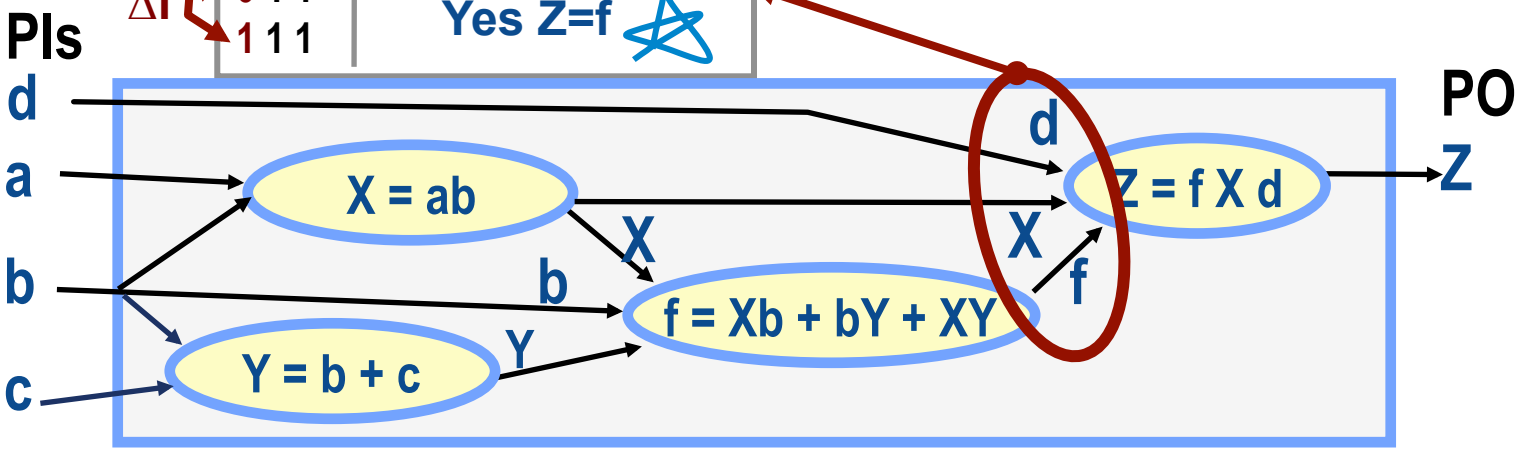


When Is Z "Sensitive" to Value of f

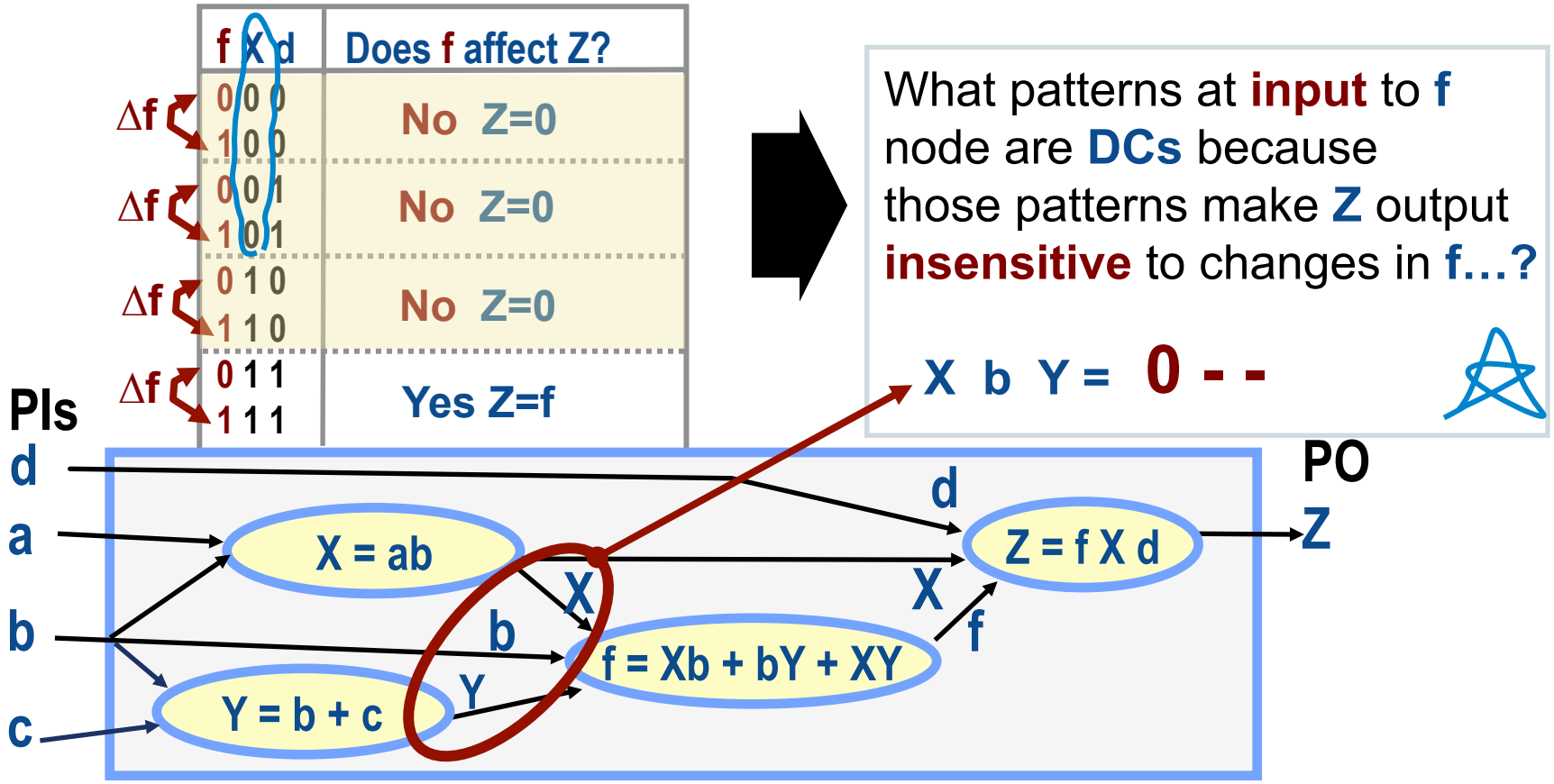
	f X d	Does f affect Z?
Δf	0 0 0 1 0 0	No Z=0
Δf	0 0 1 1 0 1	No Z=0
Δf	0 1 0 1 1 0	No Z=0
Δf	0 1 1 1 1 1	Yes Z=f

Can we use this information to find new patterns of **X b Y** to help us simplify **f** further?

YES.



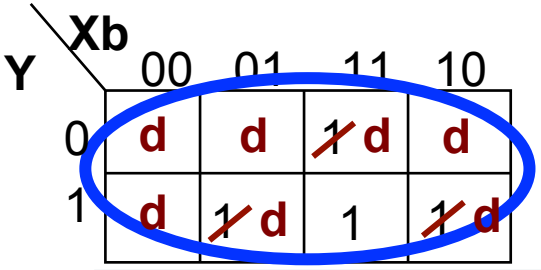
When Is Z "Sensitive" to Value of f



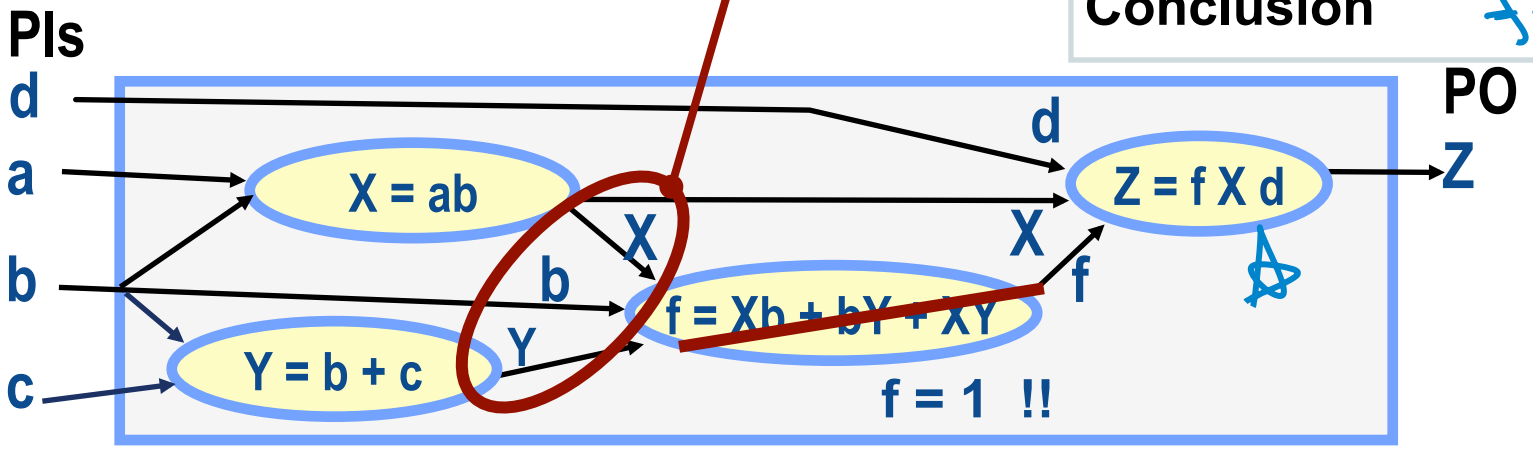
Back to Multi-level DC Tour

- **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**

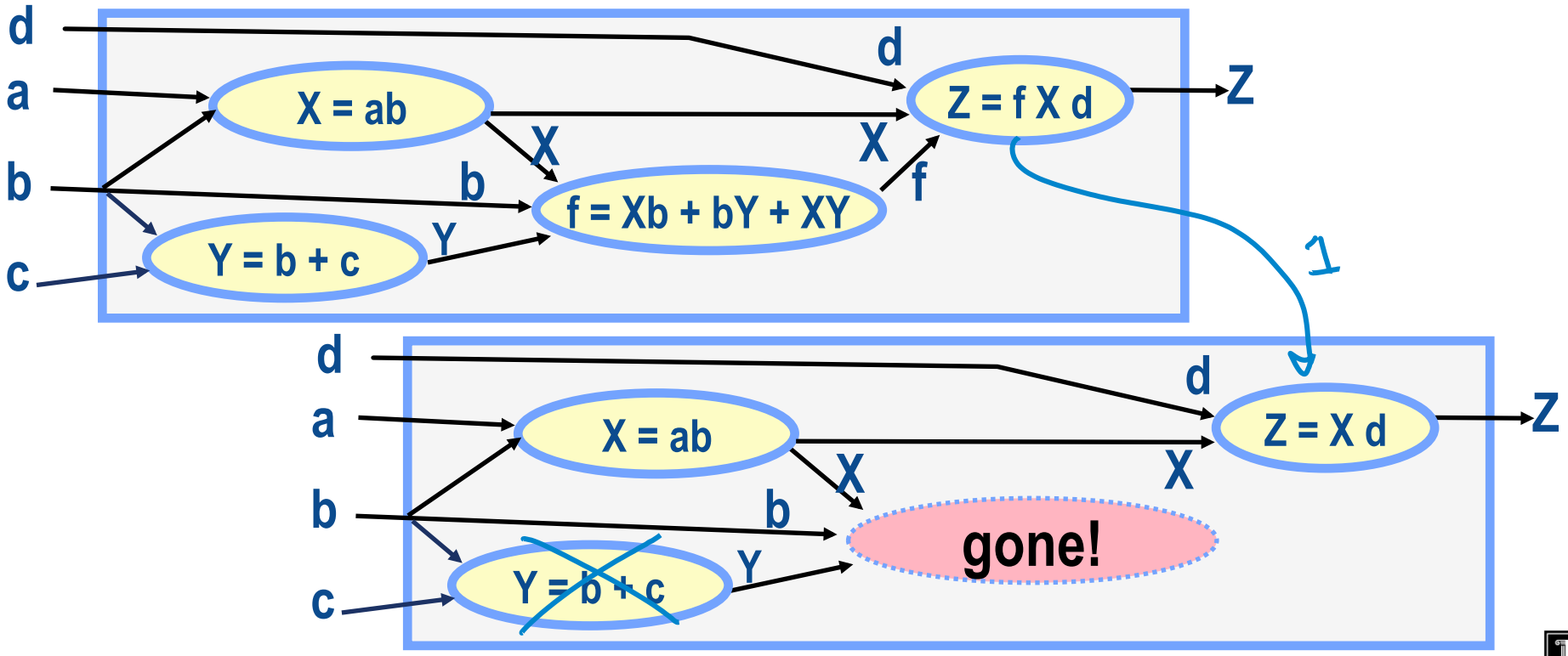


Conclusion f = 1



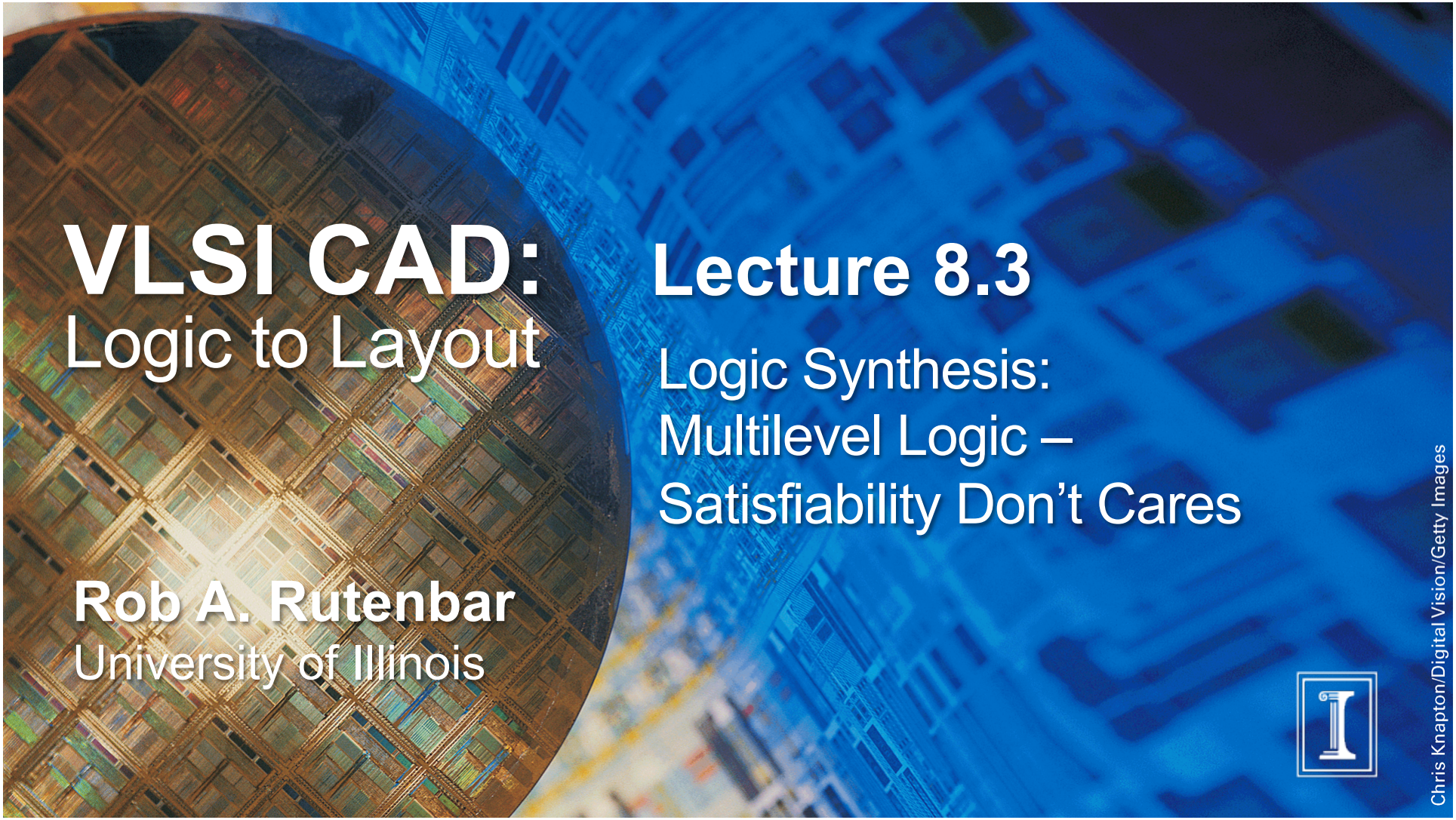
Final Result: Multi-level DC Tour

- **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

Lecture 8.3

Logic Synthesis: Multilevel Logic – Satisfiability Don't Cares

Rob A. Rutenbar
University of Illinois

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**

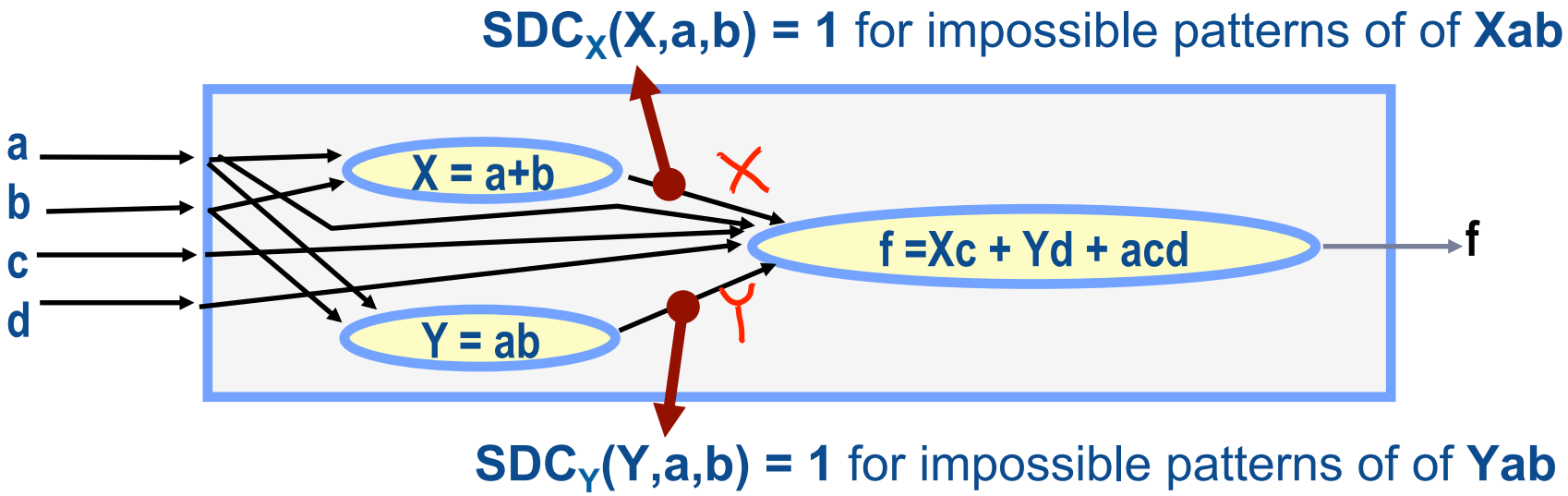
Background: Representing DC Patterns

- **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

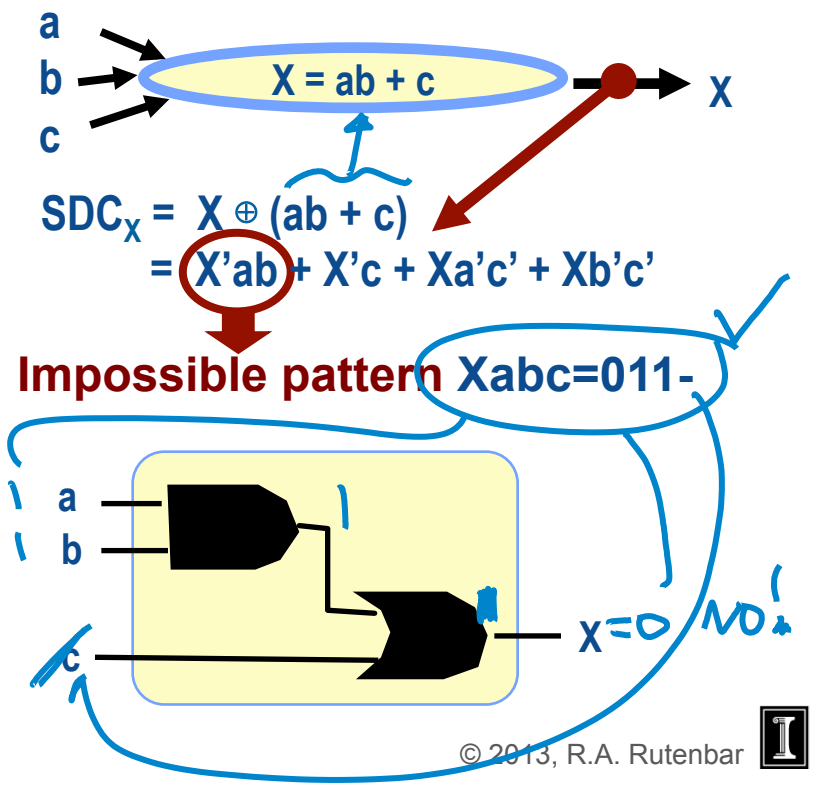
- **One SDC for every internal wire in Boolean logic network**
  - The **SDC** represents impossible patterns of **inputs to**, **and output of**, each node
  - If the node function is **F**, with inputs **(a,b,c, …)** write as: **$SDC_F(F,a,b,c…)$**



SDCs: How to Compute

- **Easy**
  - Compute an **SDC** for each output wire from each internal Boolean node
  - You want an expression that's **==1** when **X ≠ (**Boolean expression for **X)**
  - This is just **[X ⊕ (**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

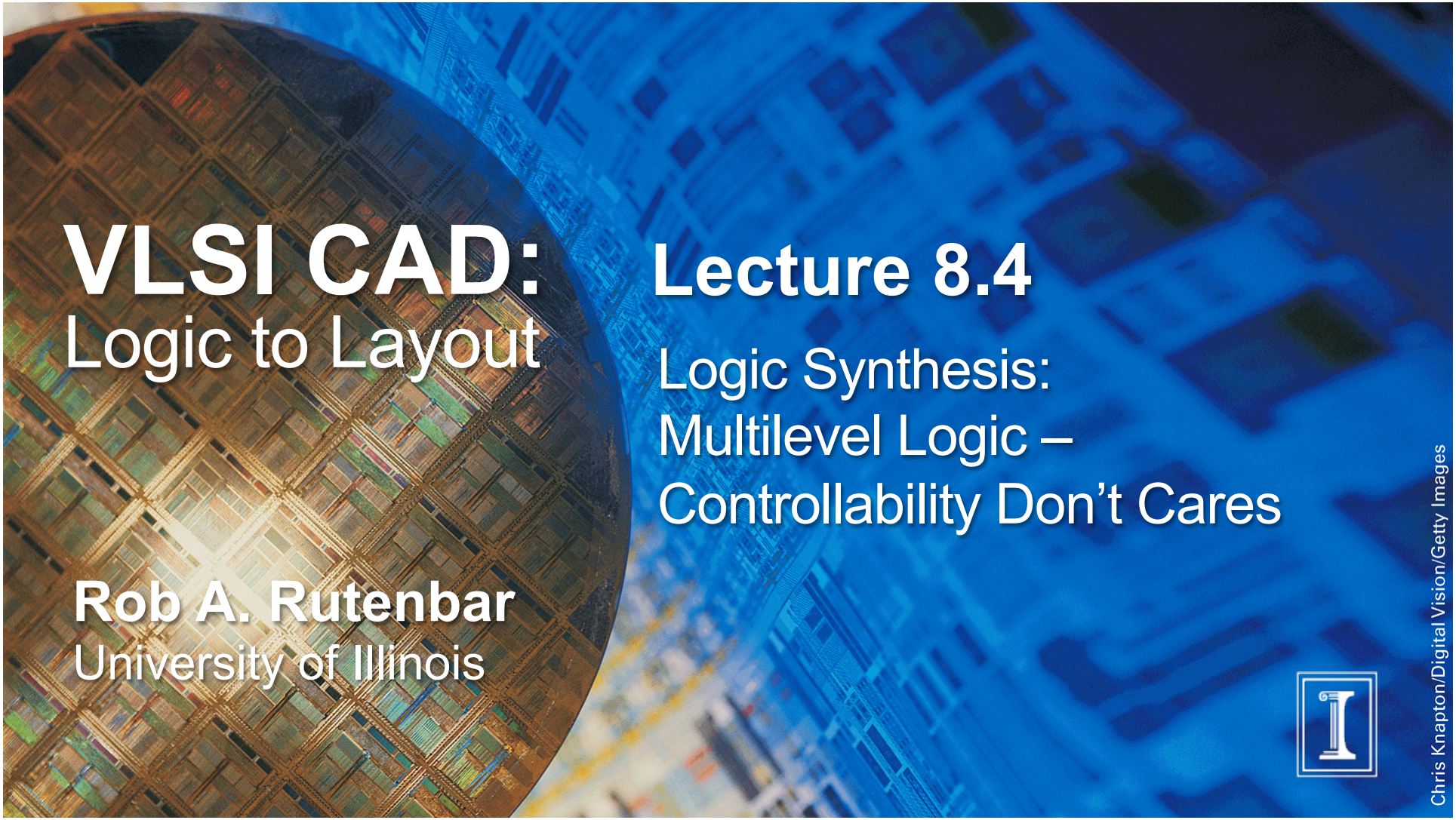
- **Example**



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

Lecture 8.4

Logic Synthesis: Multilevel Logic – Controllability Don't Cares

Rob A. Rutenbar
University of Illinois

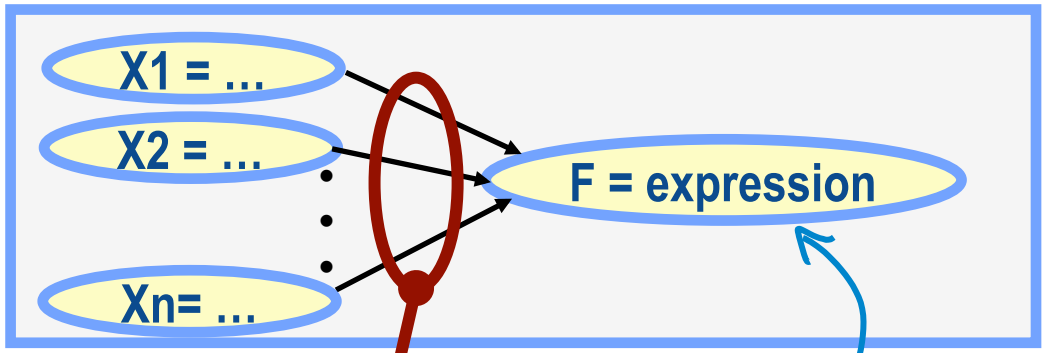
3 Types of Implicit DCs

- **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?

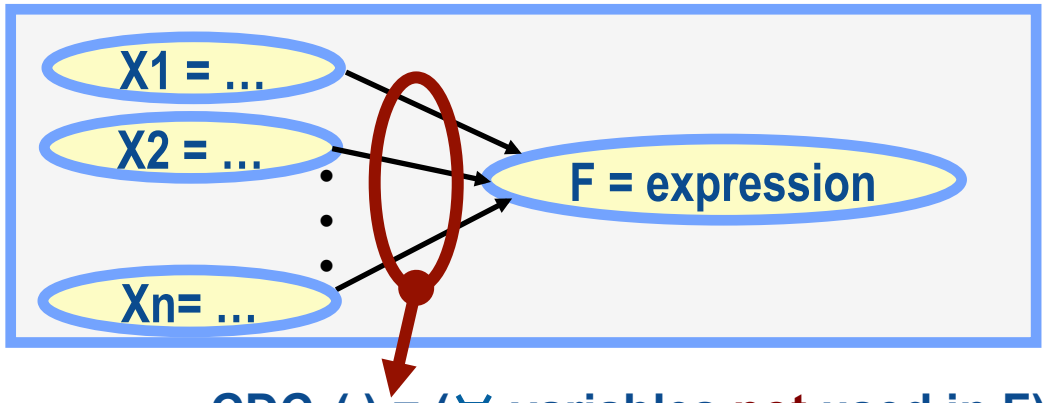
- **Surprisingly simple computational recipe for CDC_F:**
  - Get all the **SDCs** on the wires input to this node in Boolean logic network
  - **OR** together all these **SDCs**
  - **Universally Quantify** away all variables that are **NOT** used inside this node
  - **Result**: Boolean function (CDC_F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



Boolean OR

$$CDC_F( ) = (\forall \text{ variables not used in F})[\sum_{\text{input wires Xi to F}} SDC_{Xi}]$$

CDCs: Why Does This Work?

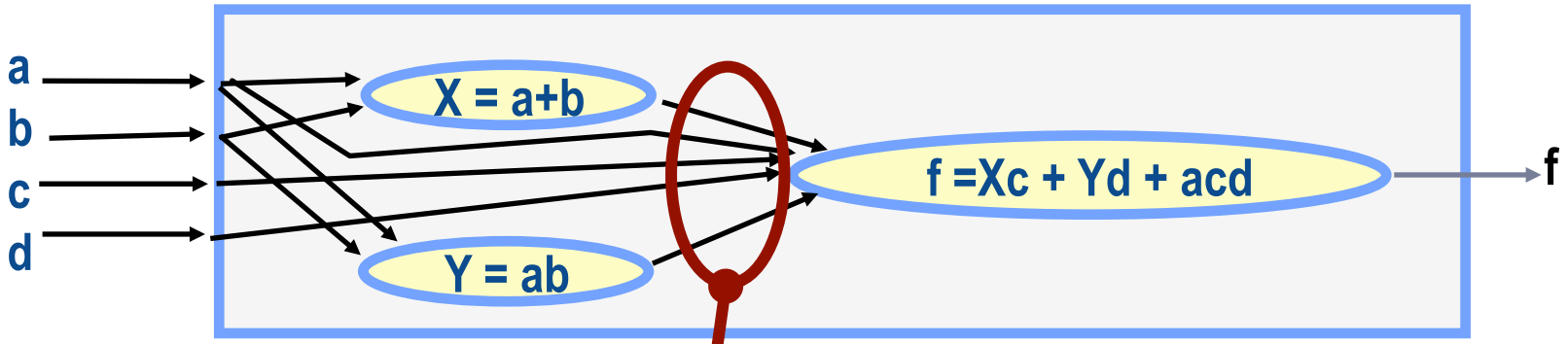


$CDC_F( ) = (\forall$ variables **not** used in F)$[\ \sum_{\text{input wires Xi to F}} SDC_{Xi}\ ]$

- **Roughly speaking…**
  - SDC_{Xi}**'s** explain all the impossible patterns involving **Xi** wires input to the **F** node
  - **OR** operation is just the "union" of all these impossible patterns involving **Xi'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 → CDC_f

- **Lets apply this computational scheme to the f node below**



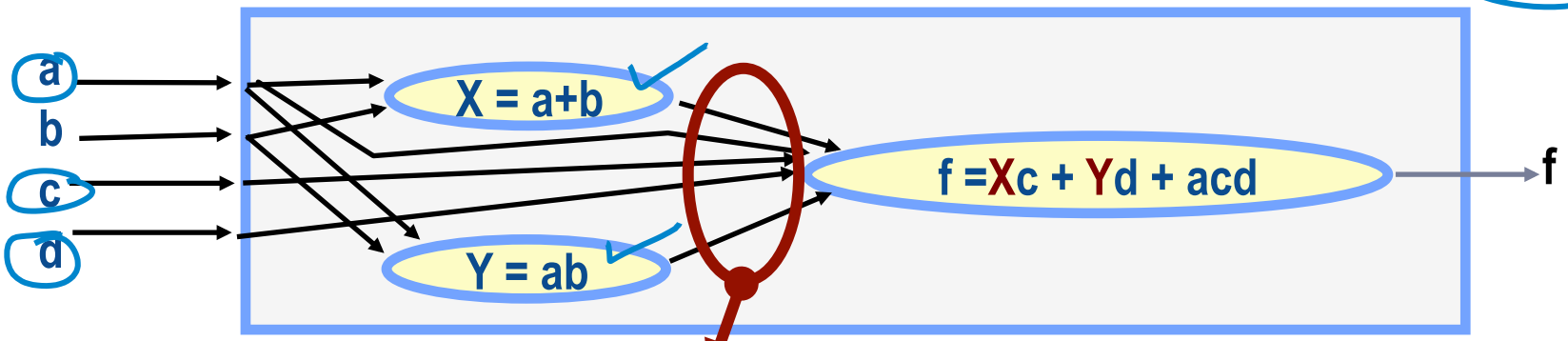
$$CDC_f = \left(\forall \text{ variables not input to f} \right) \left(\sum_{\text{inputs i}} SDC_i \right)$$

This is variable **b**

Input variables to **f** are **X, Y, a, c, d**

SDCs → 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.

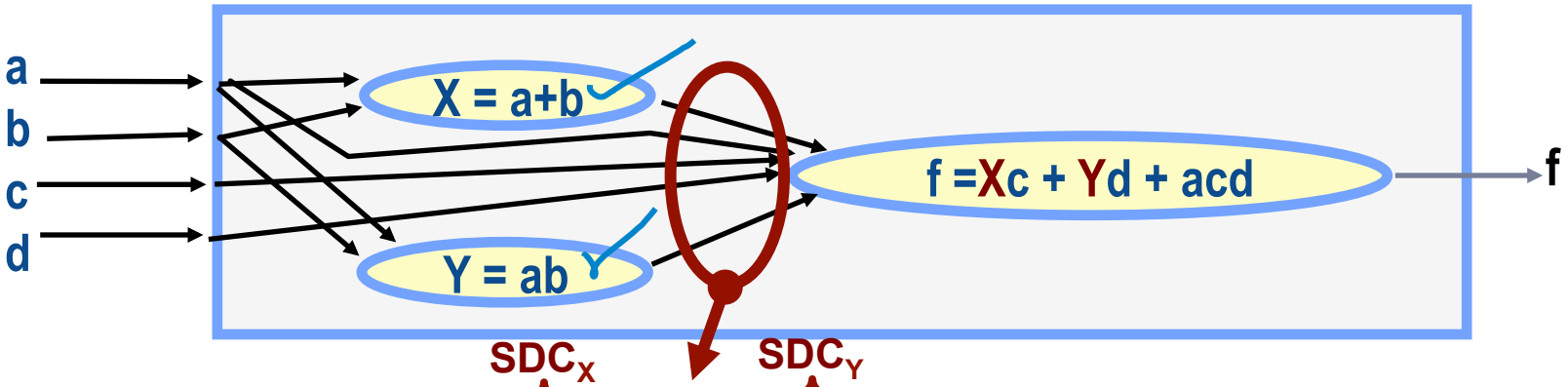


$$CDC_f = \left(\forall \text{ variables } \textbf{not} \text{ input to f} \right) \left(\sum_{\text{inputs i}} SDC_i \right)$$

Only **X, Y**

SDCs → CDCs

- **Do the (computational) Boolean algebra…**



$$CDC_f = (\forall b)(\overbrace{[X \oplus (a+b)]}^{SDC_X} + \overbrace{[Y \oplus (ab)]}^{SDC_Y}) =$$

$$([X \oplus (a+b)] + [Y \oplus (ab)])_{b=1} \cdot ([X \oplus (a+b)] + [Y \oplus (ab)])_{b=0}$$

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

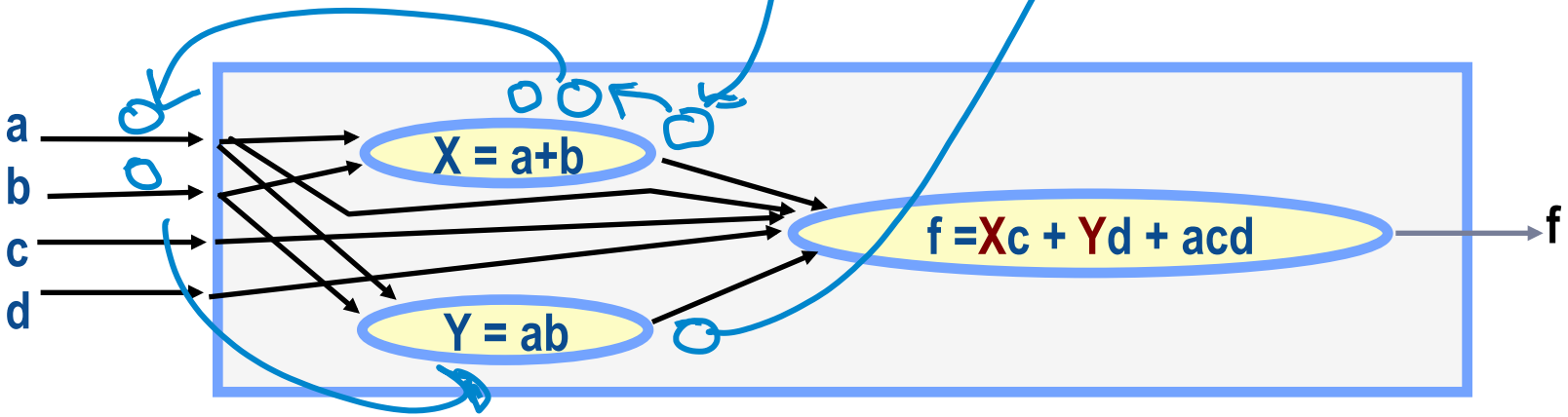
Bug fix Ya' not Y'a

CDC_f – Does It Make Sense?

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

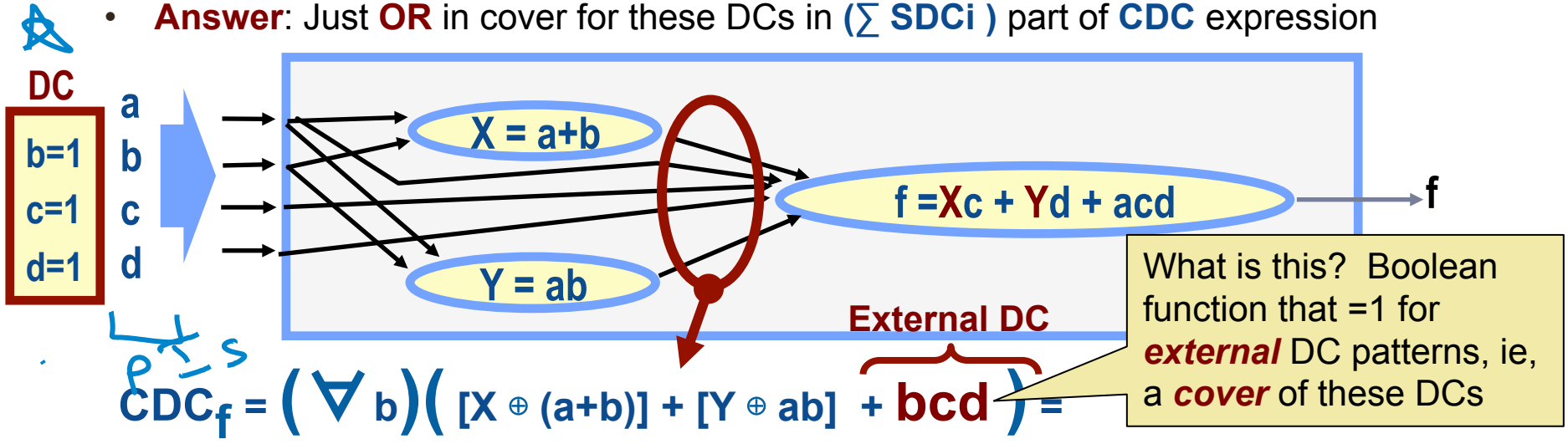
Same Bug: Ya' not Y'a

Correct!



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 (∑ SDCi) part of **CDC** expression



$$CDC_f = (\forall b)([X \oplus (a+b)] + [Y \oplus ab] + bcd) =$$

$$= [\text{ prior result } X'a + X'Y + Ya'] + (a'cdX + acdY)$$

Bug fix: Ya' not Y'a

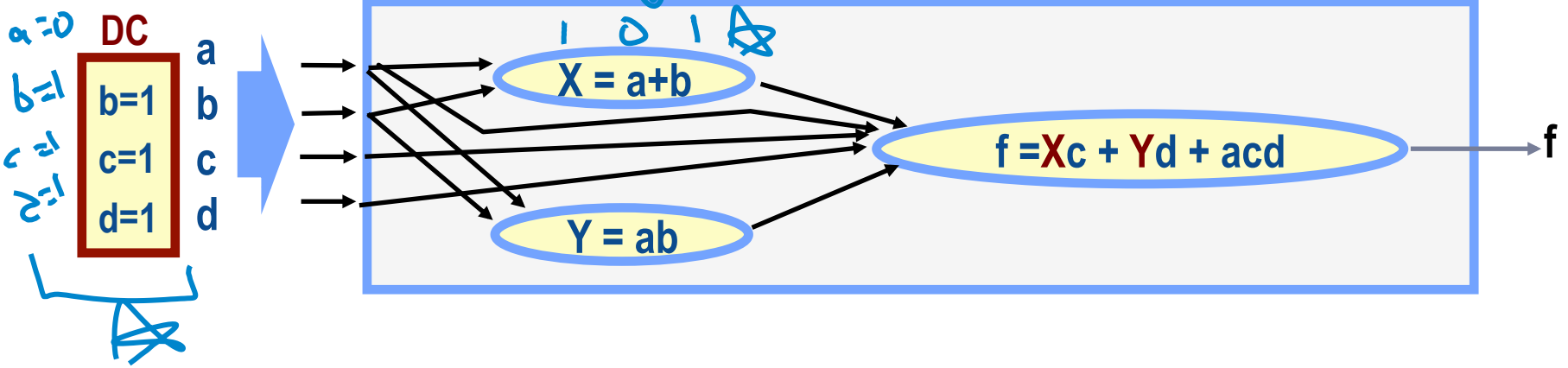
Bug fix: acdY not acdX'

External CDCs: Does It Make Sense?

Bug fix: Ya' not Y'a

Bug fix: acdY not acdX'

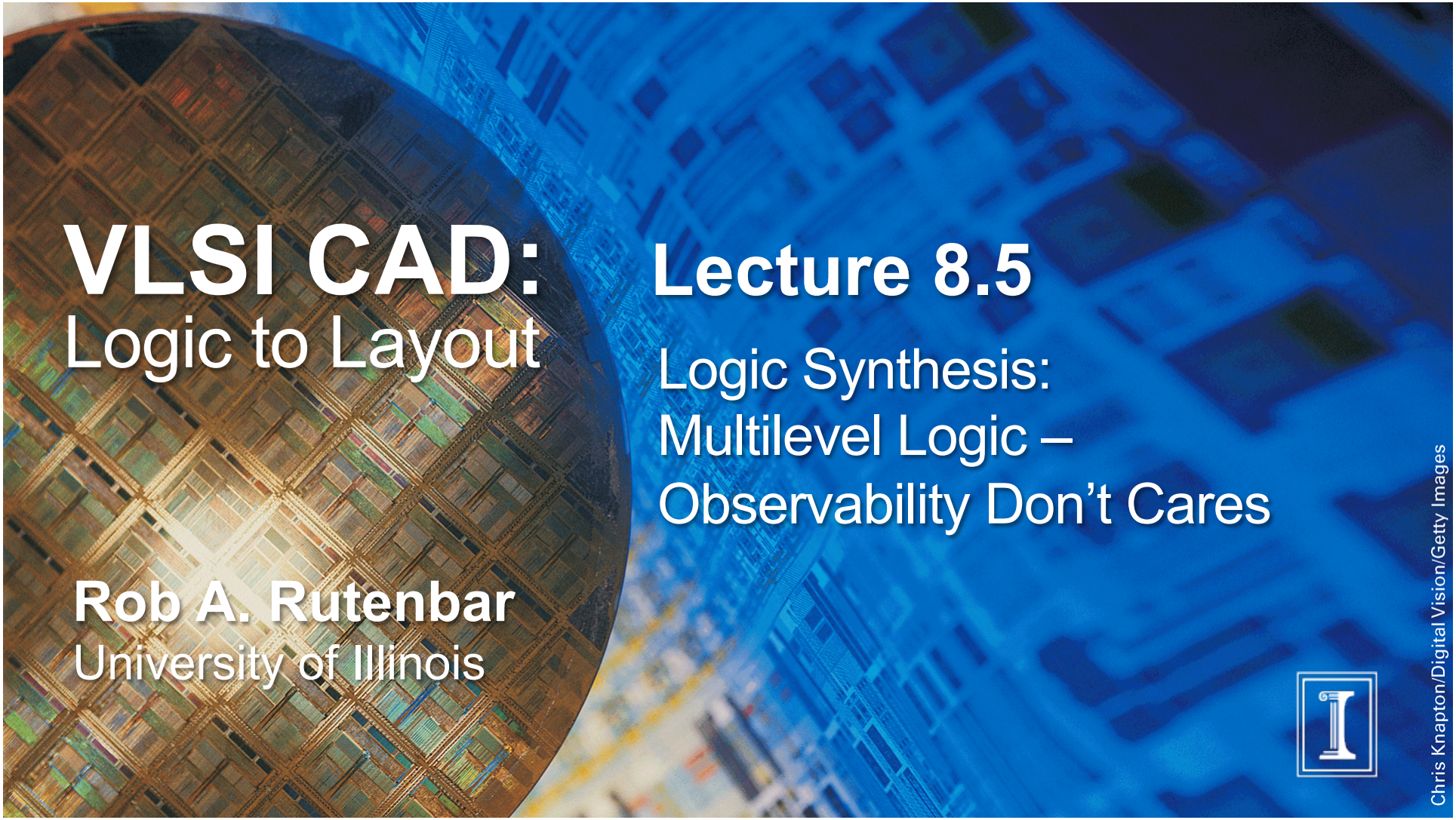
- New CDC_f = X'a + X'Y + Ya' + a'cdX + acdY impossible for f
  - a'cdX=1 → a=0 && X=1 → b must be 1 → b=1 && c=1 && d=1 → Don't Care!



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**
  - $\mathbf{CDC_F}$ derived from the structure of nodes **"in front of"** node **F**
  - We need to look at DCs that derive form 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

Lecture 8.5

Logic Synthesis: Multilevel Logic – Observability Don't Cares

Rob A. Rutenbar
University of Illinois

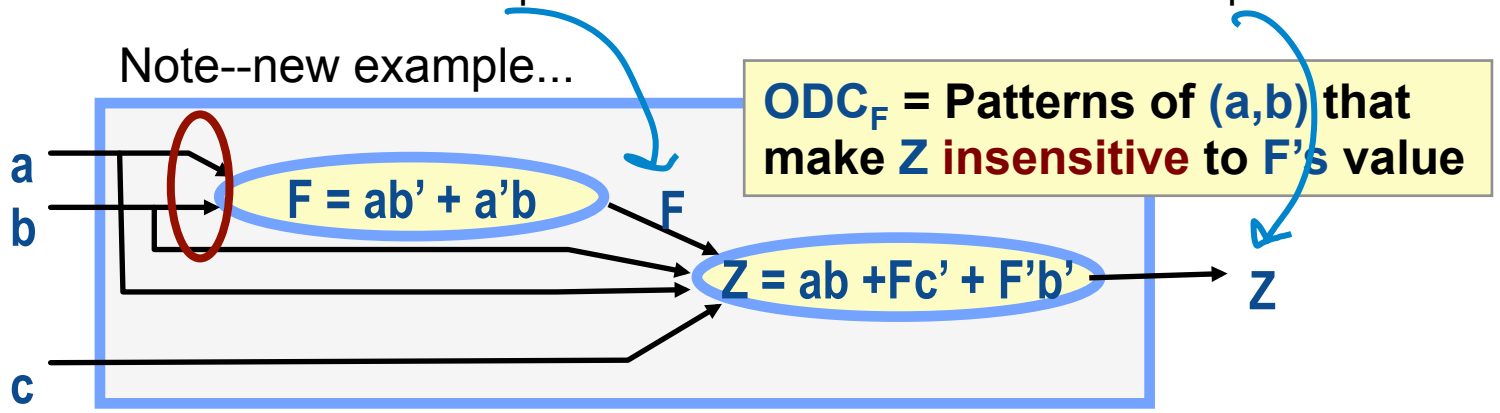
3 Types of Implicit DCs

- **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**

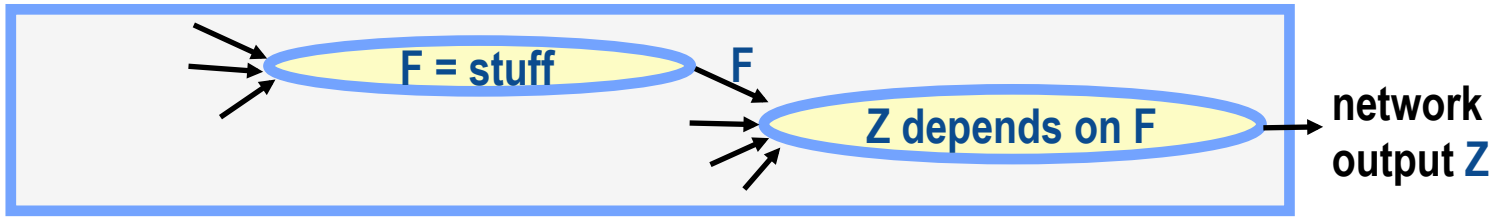
Final DCs: *Observability* Don't Cares

- **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

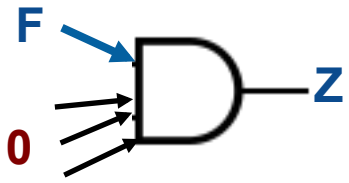


Network Output *Insensitive* to F

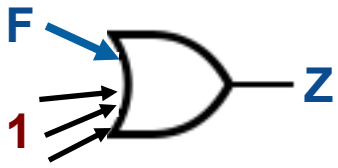
- **When is network output Z *insensitive* to internal variable F?**



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



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



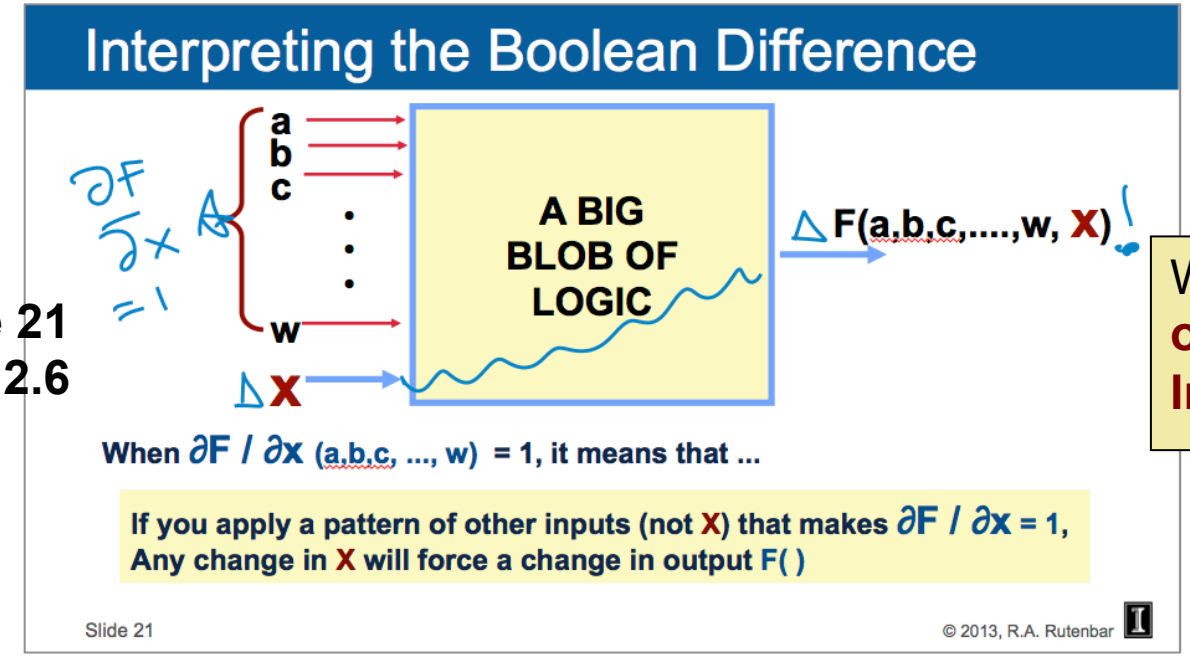
Z insensitive to **F**
if any other **input = 1**

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**

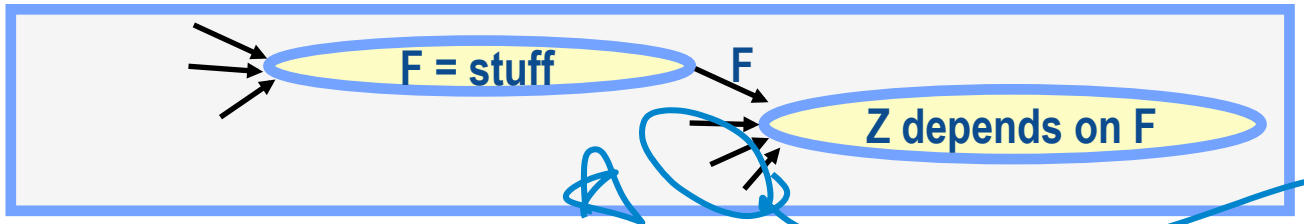
Slide 21
Lecture 2.6



We want the **opposite:** want **F Insensitive** to **X**

Z *Insensitive* to F When $\overline{(\partial Z/\partial F)}$==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**!

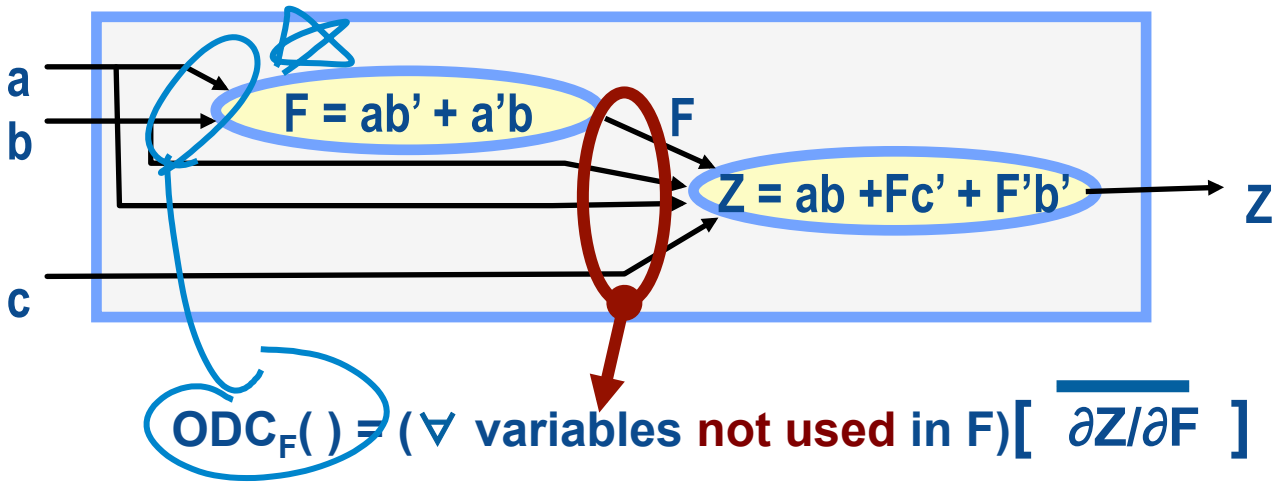
How to **find** when cofactors are the same? **Solve** for $Z(F=0) \oplus Z(F=1) == 1$

Note: $Z(F=0) \oplus Z(F=1) == 1$ same as $[\, Z(F=0) \oplus Z(F=1) \,] == 1 \rightarrow \overline{(\partial Z/\partial F)}==1$

$\partial Z/\partial F$!

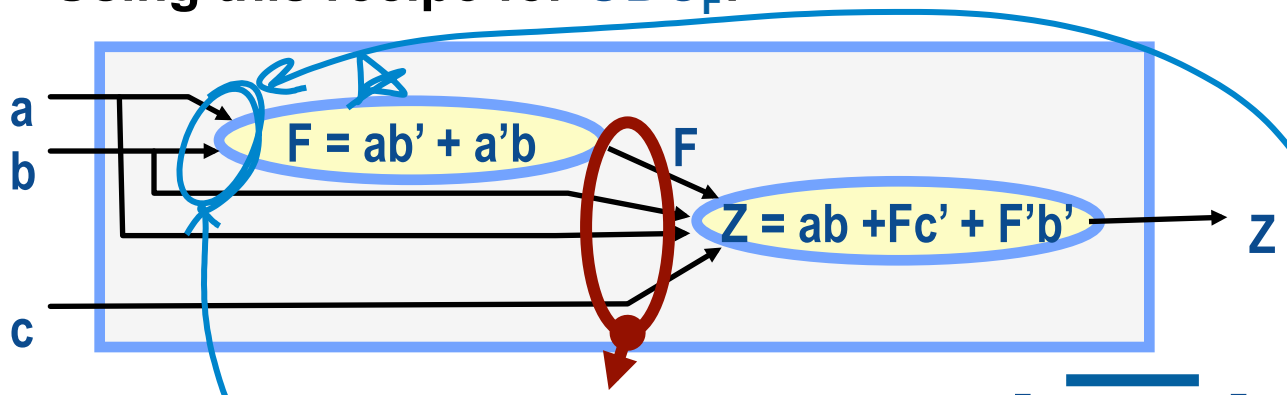
How to Get ODC Patterns that Mask Node Output?

- **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:**



$$ODC_F( ) = (\forall \text{ variables not used in F})\left[\ \overline{\partial Z/\partial F}\ \right]$$

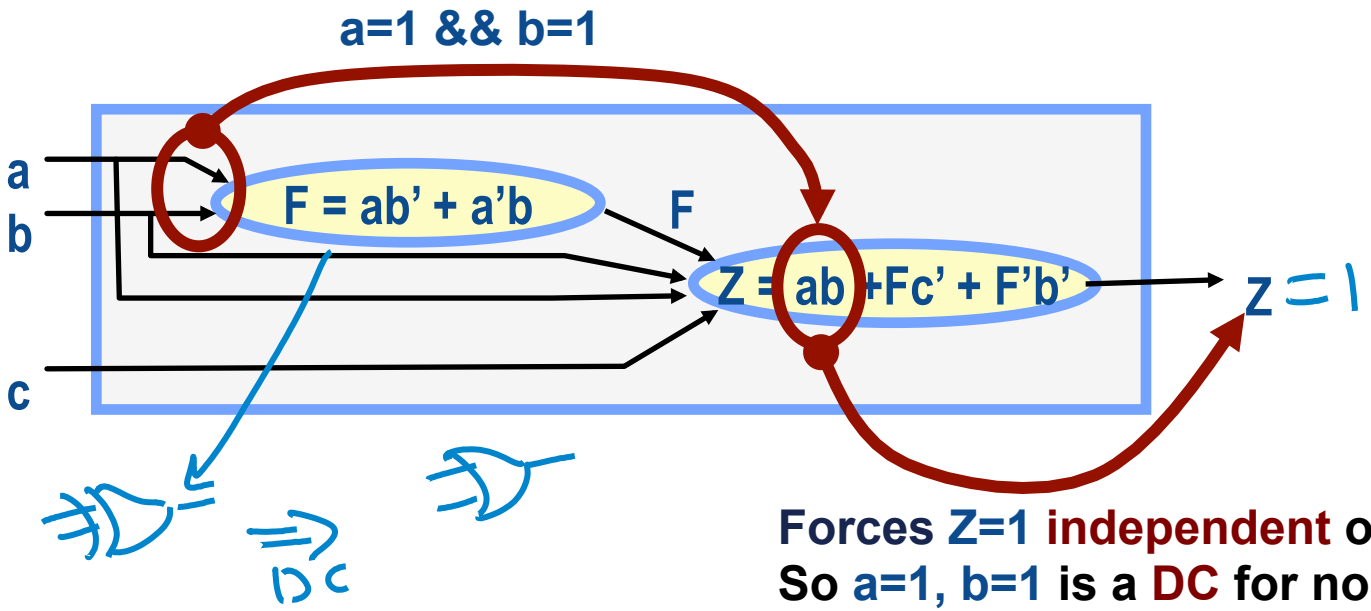
$$= (\forall\ c)\left[\ \overline{(ab+Fc'+F'b')_{F=1} \oplus (ab+Fc'+F'b')_{F=0}}\ \right]$$

$$= (\forall\ c)[\ ab + ac' + a'b'c' + a'bc\ ] = (ab)$$

Bug fix: a'b'c' not b'c'

Check: Does this ODC Make Sense?

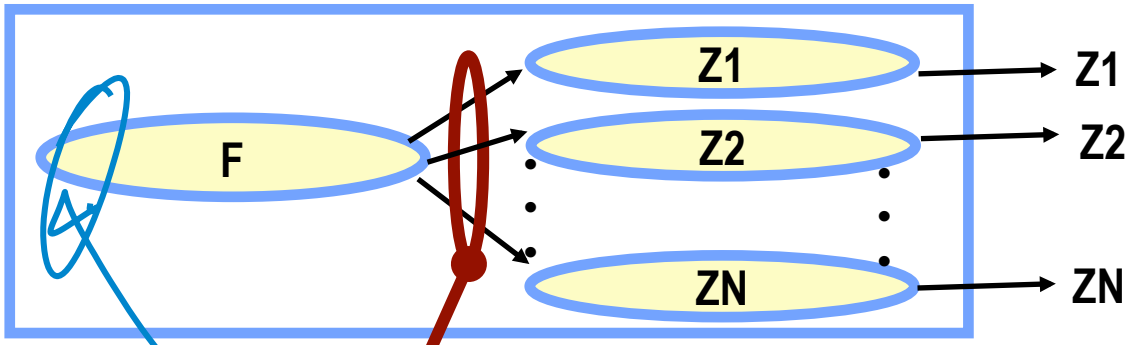
- **Yes: $ODC_F(a,b) = ab$ → a=1, b=1 masks F from affecting Z output**



Forces Z=1 independent of F
So a=1, b=1 is a DC for node F

ODCS: More General Case

- **Question: what if many network outputs?**
  - **Answer**: Only patterns that are unobservable at **ALL** outputs can be **ODCs**



AND all N differences, for each output Zi

General recipe: $ODC_F = (\forall$ variables **not** input to **F**$)[\prod_{\text{outputs i}} \overline{\partial Zi / \partial F}]$

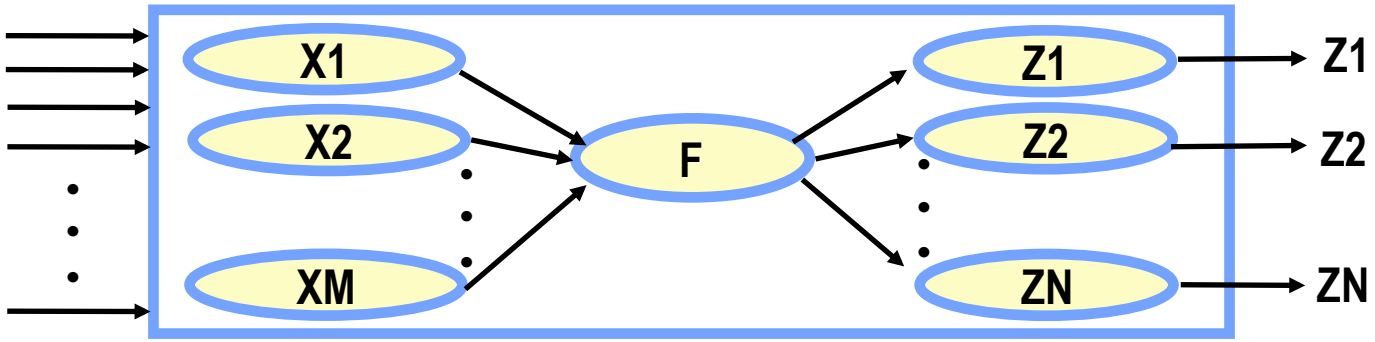
= patterns that make **F** unobservable at **ALL Zi** outputs

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 **∂Zi /∂F's** on all outputs connected to **F**

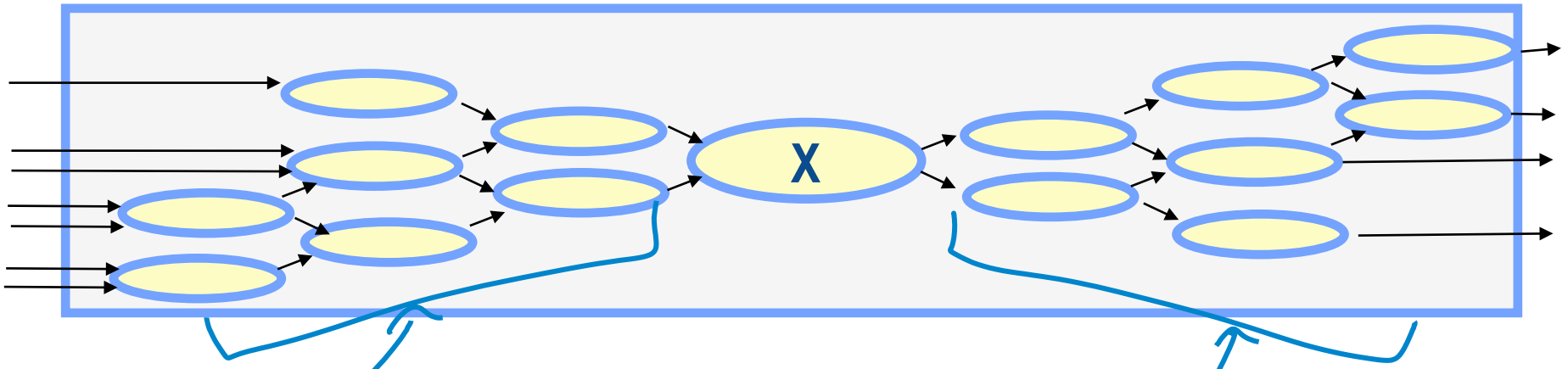
- **CDCs + ODCs give the "full" don't care set used to simplify F**
  - With these patterns, you can call something like ESPRESSO to simplify **F**

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 outout
  - 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!