Lecture Outline

Reminders to self:

- ❑ Turn on lecture recording to Cloud
- ❑ Turn on Zoom microphone

- Last Lecture
  - Finished full-adder
  - Ripple carry adder and subtracter
  - Adder with carry look ahead
- Today's Lecture
  - A couple more slides on adders with carry look ahead
  - A bit more on K-maps
  - Start multi-level logic

Handouts and Announcements

- Announcements
  - Homework Problems – no new assignment
  - Homework Reminders
    - Problems 5-1 and 5-2
    - Due: 11:25am Monday 2/6
  - Participation Quiz 5
    - Based on today's lecture. 15min limit after you start.
    - Available 12:25pm today, due 12:25pm tomorrow, but also available 24hrs more with late penalty
  - Read for Monday: pages 204-214

Handouts and Announcements

- Announcements
  - Mini-Exam 2 Reminder
    - Available 5pm Monday 2/6 through 5:00pm Tuesday 2/7
    - Due in Carmen PROMPTLY at 5:00pm on 2/7
    - Designed to be completed in ~36 min, but you may use more
    - When planning your schedule:
      - I recommend building in 10-15 min extra
      - To allow for downloading exam, signing and dating honor pledge, saving solution as pdf, and uploading to Carmen
    - I also recommend not procrastinating
  - Exam review topics available on Carmen
  - Sample Mini-Exams 1 and 2 from Au20 also available

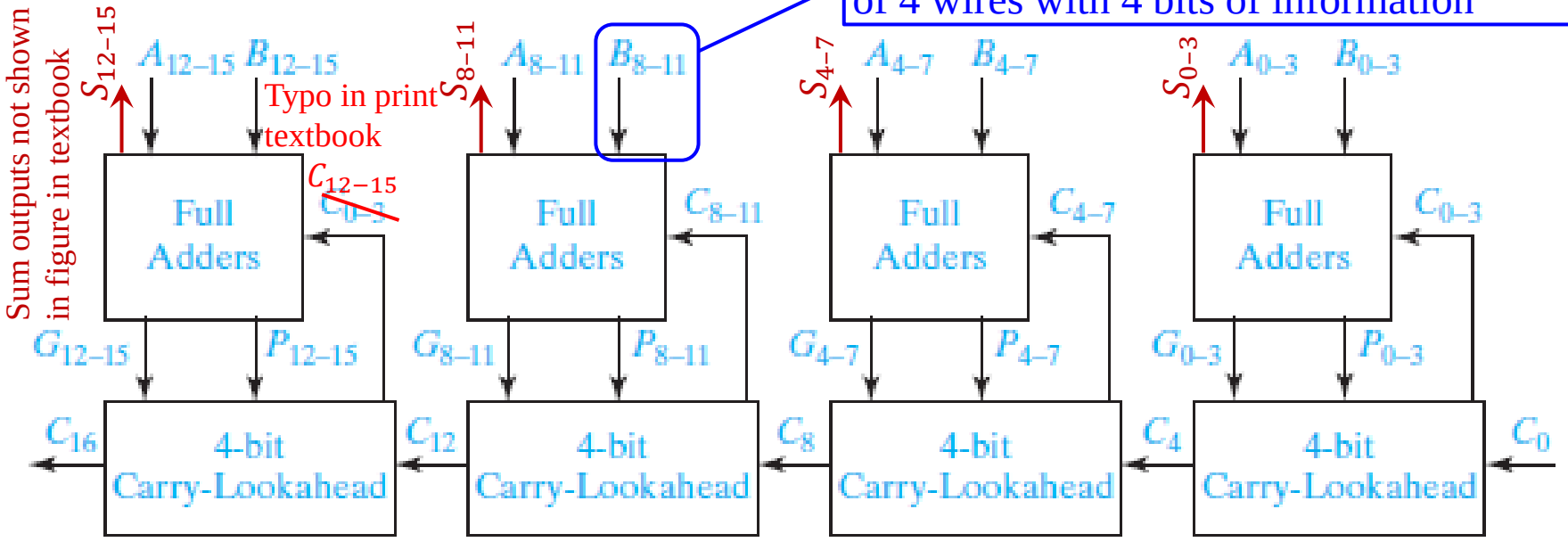
16-bit Carry-Lookahead Adder

Where we left off last lecture

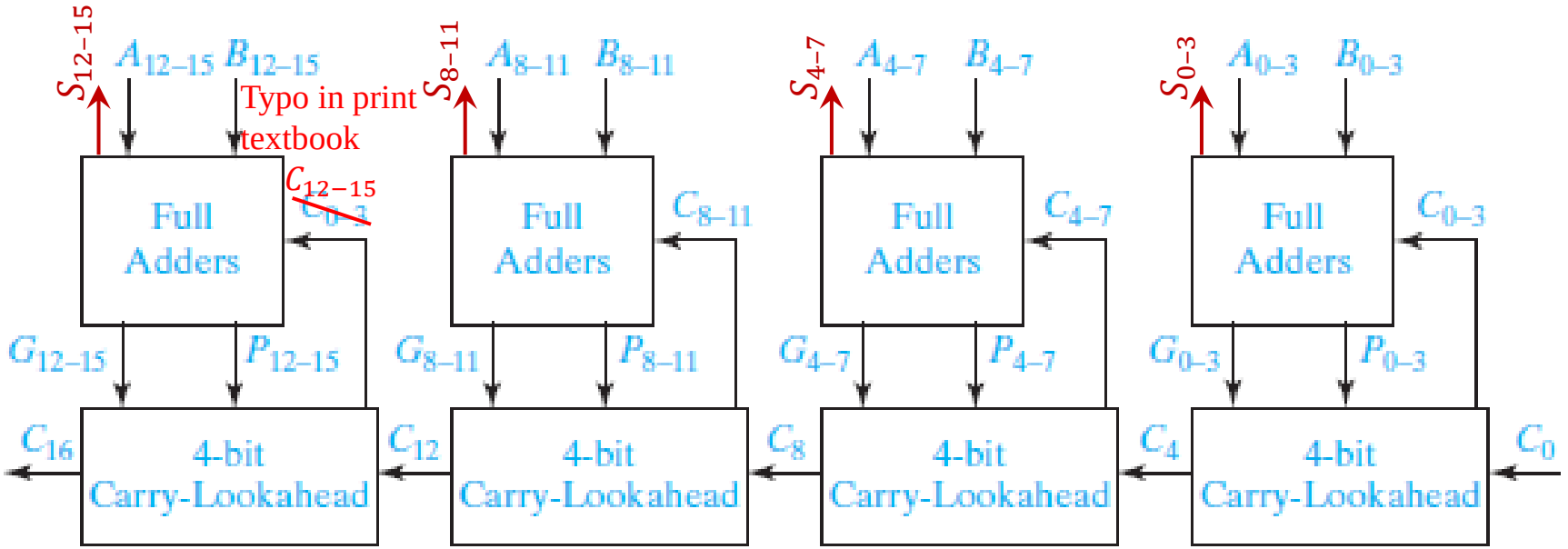
- If more than four bits need to be added
  - In principle the carry-lookahead circuit can be expanded
  - But each additional bit means number of inputs to AND and OR gates increases
    - Number of inputs is known as "fan-in"
    - Practical limits (from transistor-level design of gates) set a maximum fan-in
  - In practice, can cascade the 4-bit carry-lookahead circuits
  - A 16-bit adder shown here

Group of wires = Bus

Observe notation: Now represents a "bus" of 4 wires with 4 bits of information



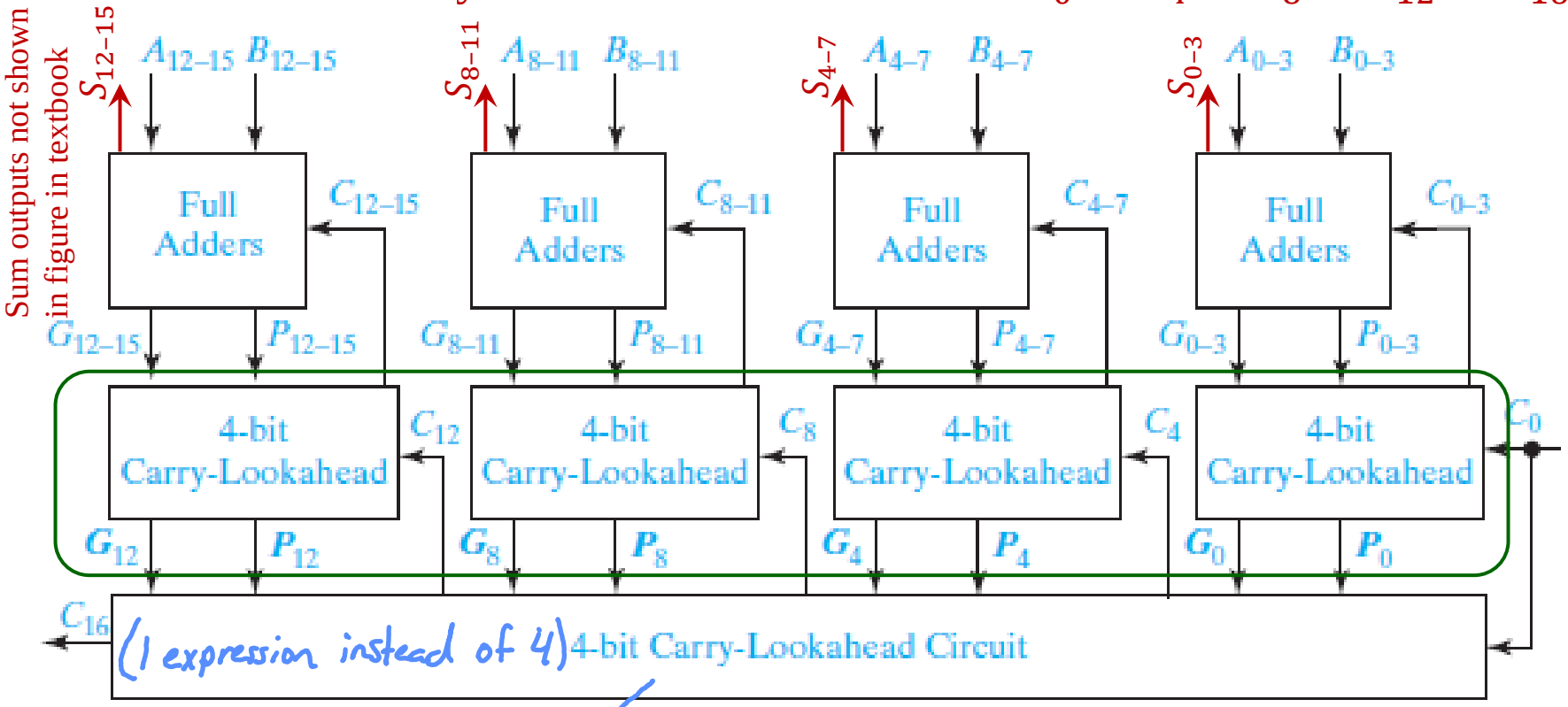
16-bit Carry-Lookahead Adder



- But now there is ripple delay cascading through the Carry-Lookahead circuits: $C_0 \Rightarrow C_4 \Rightarrow C_8 \Rightarrow C_{12} \Rightarrow C_{16}$
- It is still an improvement over brute force 16 bit ripple adder
  - Brute force 16 bit ripple adder: 32 gate delays for C_{16} to settle
  - 16 bit adder shown here: 8 gate delays
- Additional improvement possible: additional level of Carry-Lookahead

16-bit Carry-Lookahead Adder

Additional level of Carry-Lookahead added to handle $C_0 \Rightarrow C_4 \Rightarrow C_8 \Rightarrow C_{12} \Rightarrow C_{16}$



This is same Carry-Lookahead circuit as before

But first level of Carry-Lookahead modified to output G, P and C_i as shown, but not the carries to pass forward

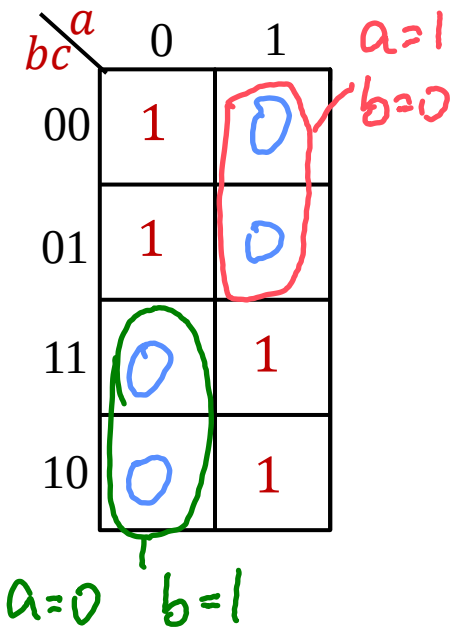
POS from K-Map

To get reduced POS form of function F from K-Map

1. Group the zeros (Maxterms) in K-Map
2. Use those groups to write $\bar{F}$ in SOP form
3. Use DeMorgan's to convert to F in POS form

Example: $F(a,b,c) = \sum m(0,1,6,7)$

Find reduced POS form of F



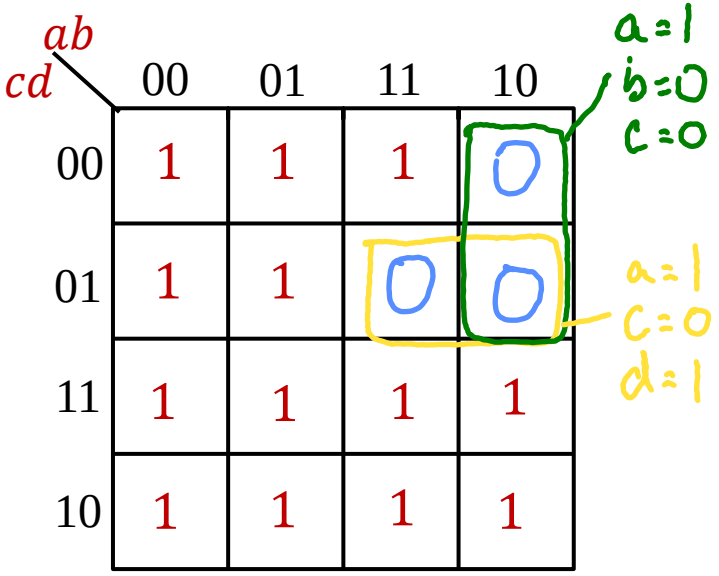
$\bar{F} = a\bar{b} + \bar{a}b$

$$F = \overline{a\bar{b} + \bar{a}b} = \left(\overline{a\bar{b}}\right)\left(\overline{\bar{a}b}\right) = (\bar{a} + b)(a + \bar{b})$$

POS from K-Map

Example: $F(a,b,c,d) = \sum m(0,1,2,3,4,5,6,7,10,11,12,14,15)$

Find reduced POS form of F



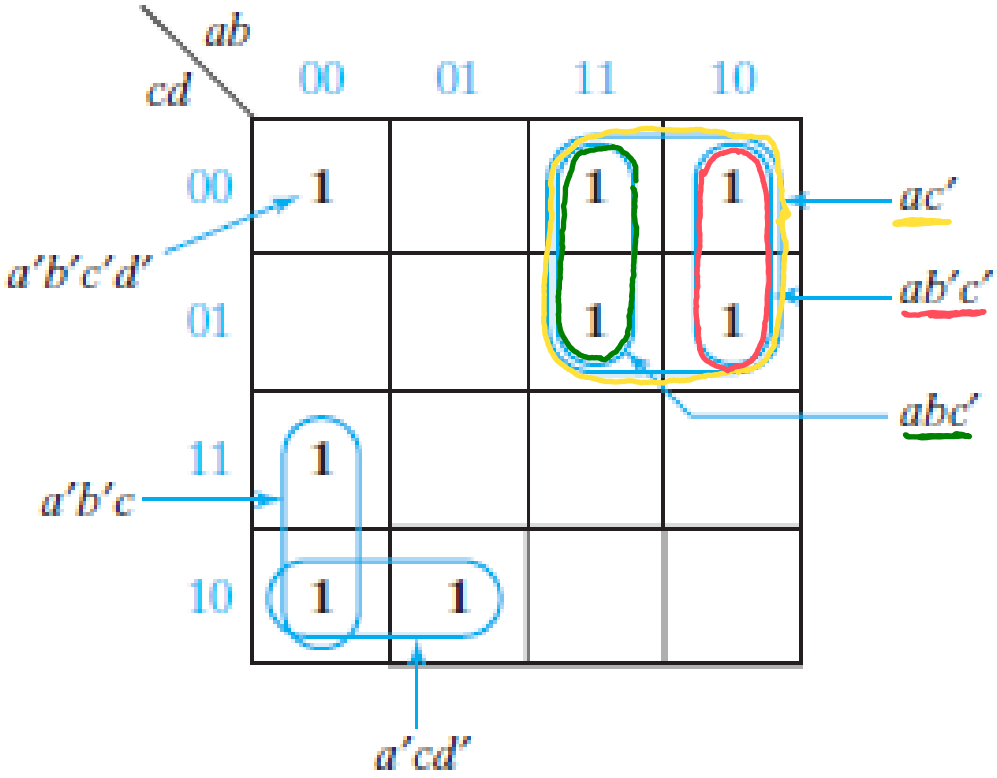
$F' = ab'c' + ac'd$

$F = (a'+b+c)(a'+c+d')$

$F = [ab'c' + ac'd]' = (ab'c')'(ac'd)' = (a' + b + c)(a' + c + d')$

Prime Implicants

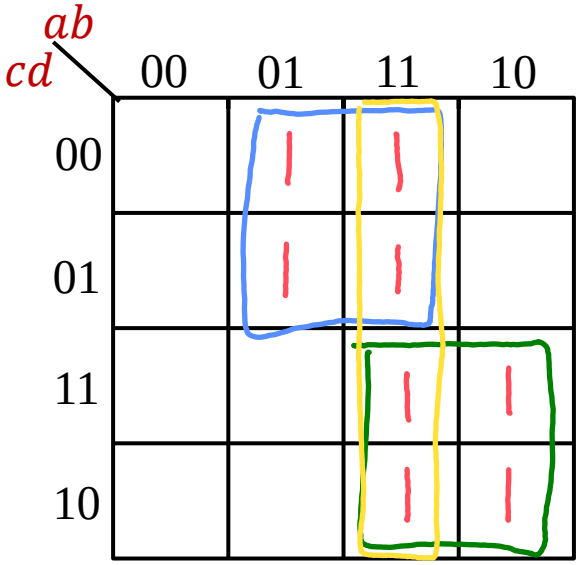
- The product terms that result from groups in K-maps are known as "implicants" of the function
- An implicant is also a "prime" implicant if it cannot be combined with another term to eliminate a variable



- $a'b'c$, $a'cd'$, and ac' are prime implicants
- Not prime implicants:
  - $a'b'c'd'$ → can be grouped into $a'b'd'$ or $b'c'd'$ prime
  - abc' and $ab'c'$ → can be grouped into ac' prime
- A SOP expression containing a term that is not a **prime** implicant is not minimal

Prime Implicants

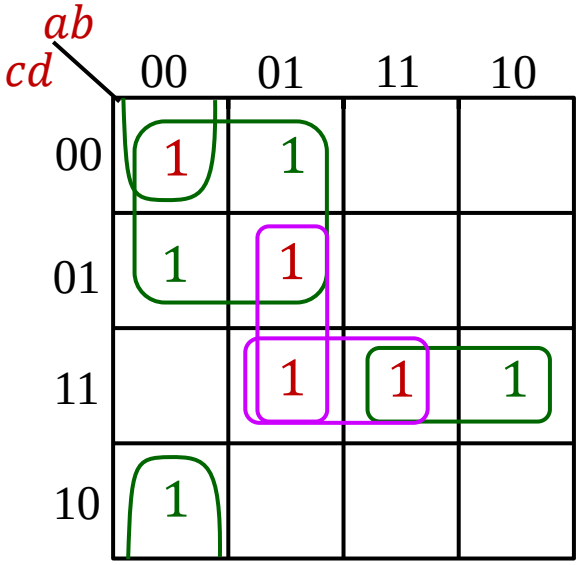
- A SOP expression containing a term that is not a prime implicant is not minimal
- Redundant terms: It is possible for all terms to be prime implicants but the function to not be minimal



- bc', ac, and ab are prime implicants
- But $F = bc' + ac + ab$ is not minimal
- Prime implicant ab is redundant
- Minimal $F = bc' + ac$

Essential Prime Implicants

- If a minterm is covered by only one prime implicant, that prime implicant is an "essential" prime implicant
- Essential prime implicants must be included in the minimal SOP expression



- All prime implicants drawn
- Red 1s covered by more than one prime implicant → not all of those implicants are essential
- Green 1s covered by only one prime implicant → those prime implicants are essential
- Essential prime implicants, drawn as green groups, must be included in SOP expression
- Only one of the two magenta groups should be included (either one is OK - expression not unique)

$$F = a'c' + acd + a'b'd' + a'bd$$

$$F = a'c' + acd + a'b'd' + bcd$$

Essential Prime Implicants
Single Output

Procedure

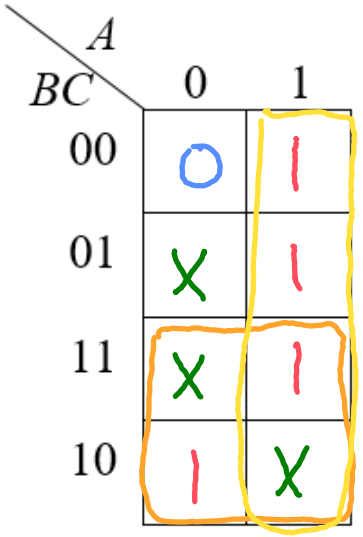
1. Choose a minterm (a 1, not an X) that has not yet been covered
2. Find all 1's and X's adjacent to that minterm (check the n adjacent squares on an n-variable K-map
3. If a single prime implicant covers the minterm and all adjacent 1's and X's then that term is an essential prime implicant → include it
4. Repeat steps 1, 2, and 3 until all essential prime implicants have been included
5. For the remaining 1's
   - Find a minimum set of prime implicants that covers them
   - If there is more than one such set, choose a set with a minimum number of literals (minimize number of gate inputs)

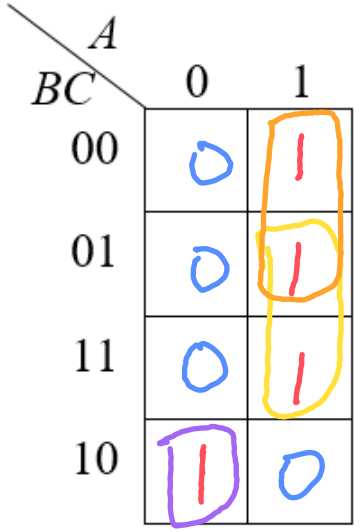
More about Don't Cares

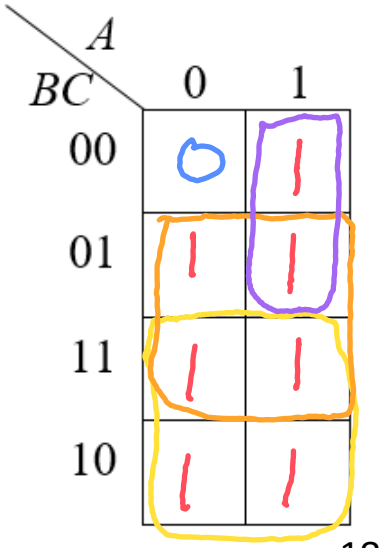
ECE2060

$F(A,B,C) = \sum m(2,4,5,7) + \sum d(1,3,6)$

A	B	C	Optimal	Ignoring X	Grouping all X
0	0	0	0	0	0
0	0	1	X	0	1
0	1	0	1	1	1
0	1	1	X	0	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	X	0	1
1	1	1	1	1	1







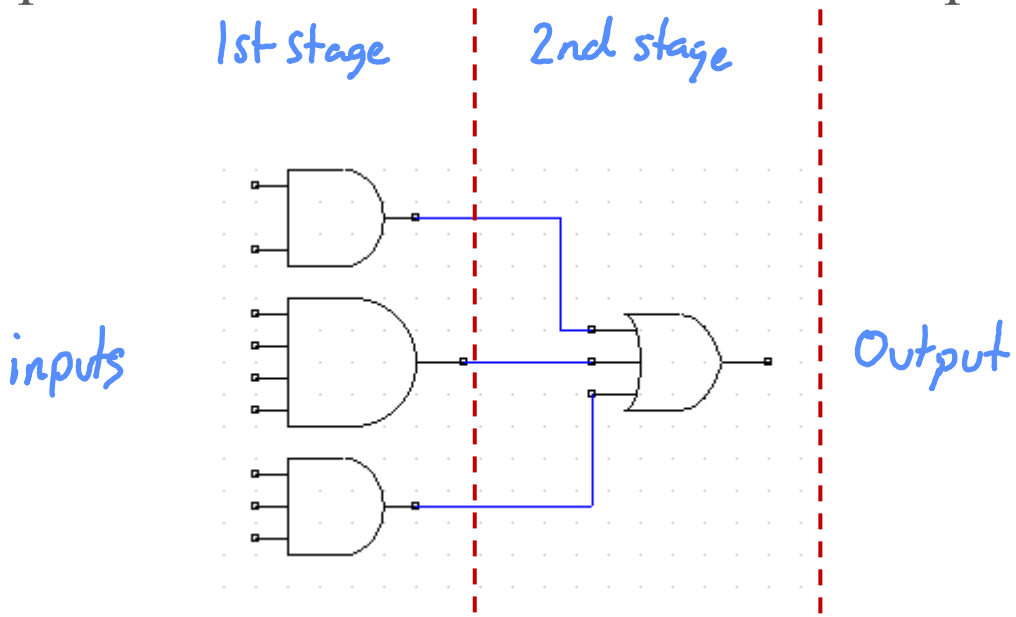
Ignoring don't cares: $F = AB' + AC + A'BC'$ 4 gates, 10 inputs

Optimal: $F = A + B$ 1 gate, 2 inputs

Grouping all X. $F = A + B + C$ 1 gate, 3 inputs

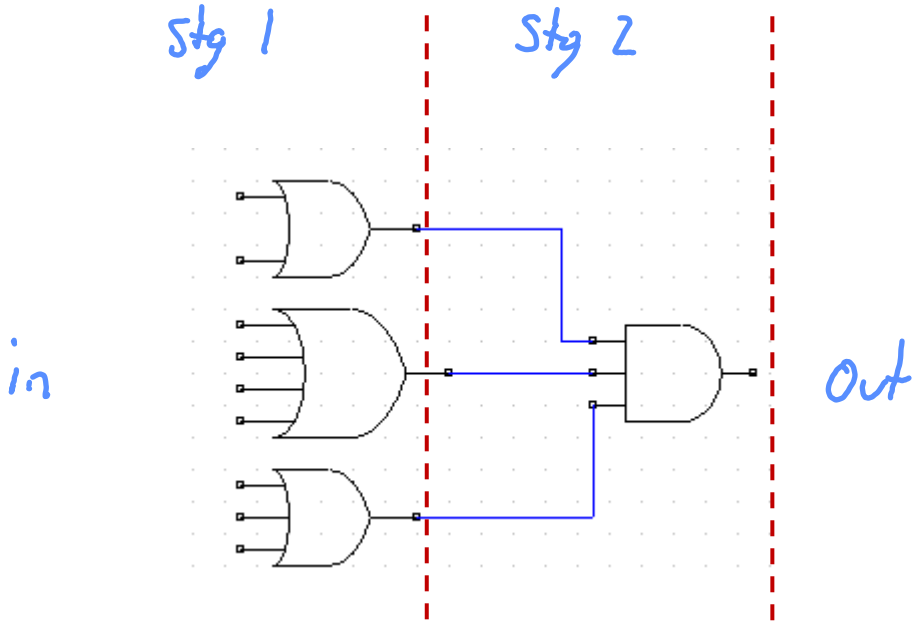
Multi-Level Logic

- Also known as Multi-Stage Logic
- **Two-level logic**
  - Sum of Products, or
  - Product of Sums
- SOP: Inputs → AND → OR → Output



Two-level Logic

- POS: Inputs → OR → AND → Output

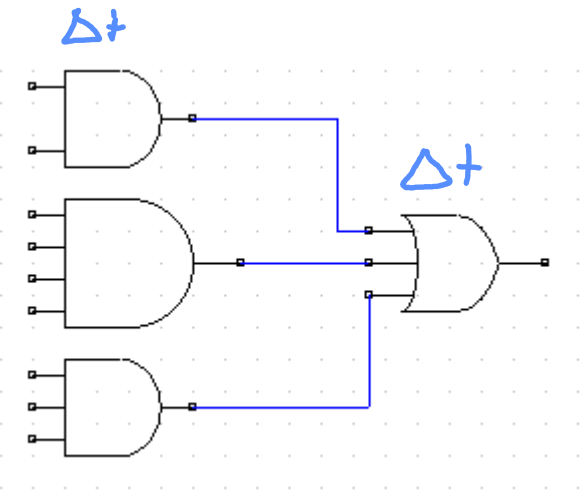


- K-Maps → SOP → 2-stage AND→OR

Two-level Logic

Primary reasons for SOP 2-Stage Design

1. Speed
   - Each "stage" has a delay (gate-delay)



   - 2-stage is the shortest for AND/OR/NOT implementation of "arbitrary" logic functions

Two-level Logic

Primary reasons for SOP 2-Stage Design

2. Regularization

- All inputs into AND gates
- All AND gates go into OR gates
- All outputs come from OR gates
- On an IC, can make:
  - An array of AND gates with possible connections
  - Another array of OR gates with possible connections from AND array
- Make input & AND-OR connections to create a logic function
- PLA : **P**rogrammable **L**ogic **A**rray
- FPGA : **F**ield **P**rogrammable **G**ate **A**rray

Multi-Level Logic

Terminology:

- **Number of levels:** maximum number of gates cascaded in series between input and output
- **AND-OR circuit:** a two-level circuit composed of a level of AND gates followed by an OR gate output (SOP form)
- **OR-AND circuit:** a two-level circuit composed of a level of OR gates followed by an AND gate output (POS form)
- **OR-AND-OR circuit:** a three-level circuit composed of
  - a level of OR gates
  - followed by a level of AND gates
  - followed by an OR gate output
- ETC.
- "*Circuit of AND and OR gates*" implies no particular ordering of the gates or number of levels; the output gate may be either AND or OR

Multi-Level Logic

Changing the numbers of levels:

- **AND-OR**: number of levels can usually be increased by factoring SOP expression
- **OR-AND:** number of levels can usually be increased by multiplying out some of the terms in POS expression
- **Why increase numbers of levels?**
  - There may be constraints on fan-in (num of inputs to a gate)
  - Can reduce fan-in at expense of adding levels
  - May also be able to reduce total number of gates by adding levels
  - Trade-off: More levels → longer overall gate-delay → slower digital system

Multi-Level Logic

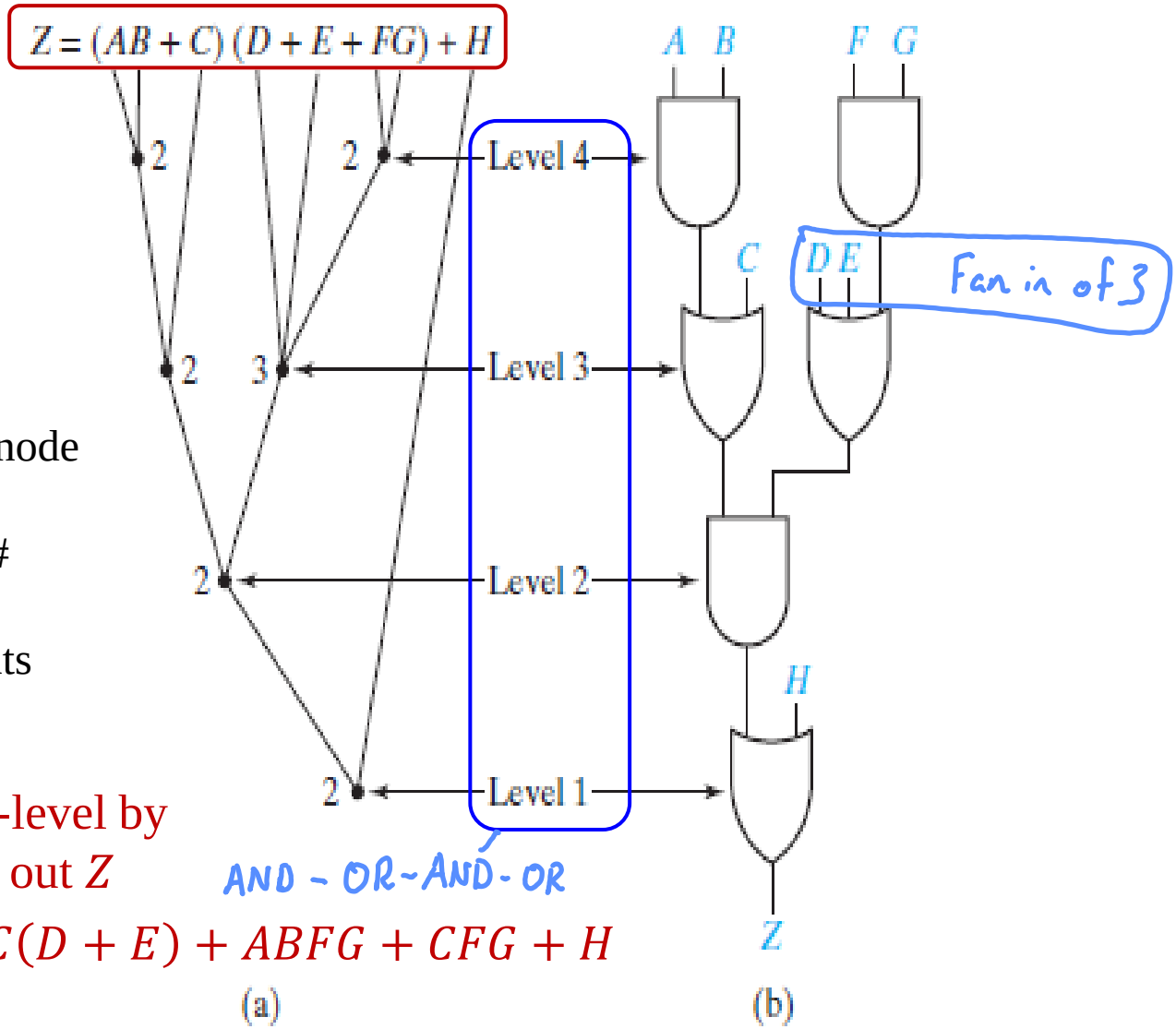


FIGURE 7-1
Four-Level Realization of Z

Tree diagram

- Each gate shown as node (6 gates)
- Nodes labeled with # gate inputs
- Total of 13 gate inputs

Can be changed to 3-level by partially multiplying out Z

$$Z = AB(D + E) + C(D + E) + ABFG + CFG + H$$

Multi-Level Logic

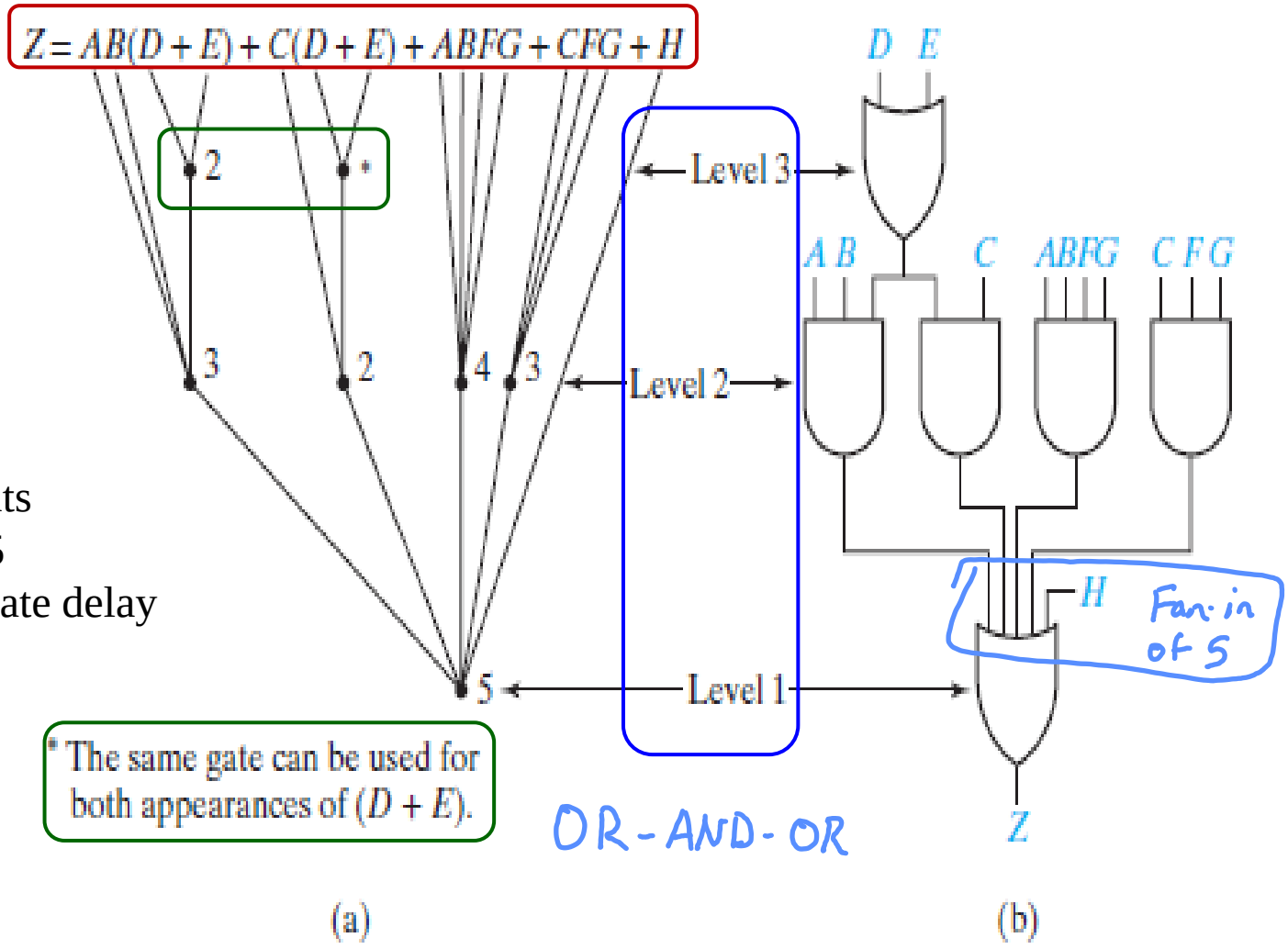


FIGURE 7-2
Three-Level Realization of Z

Tree diagram

- Still 6 gates
- Total of 19 gate inputs
- Maximum fan-in = 5
- But shorter overall gate delay

Less gates
Higher fan-in