14:332:231
DIGITAL LOGIC DESIGN

Ivan Marsic, Rutgers University
Electrical & Computer Engineering
Fall 2013

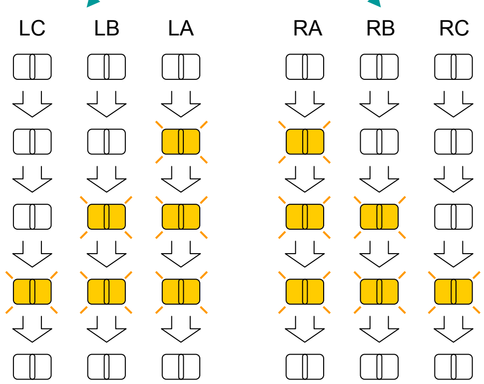
Lecture #19: Designing State Machines Using State Diagrams

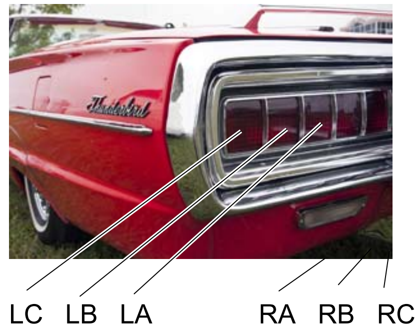
Design Steps Using State Diagrams

- Identify the inputs and outputs
- Identify the states (by their symbolic names)
- Draw the state diagram
- Analyze the state diagram for ambiguities
  - If ambiguities found, go back and modify the diagram and analyze again
    - iterative process, as the problem and solution become clearer in the designer's mind
- Derive the transition list
- Synthesize the circuit from the transition list

Example State Machine

- Controlling the tail lights of a 1965 Ford Thunderbird
- Flashing sequence for: left turn and right turn





Hazard (HAZ) lights:

All 6 lights are flashing

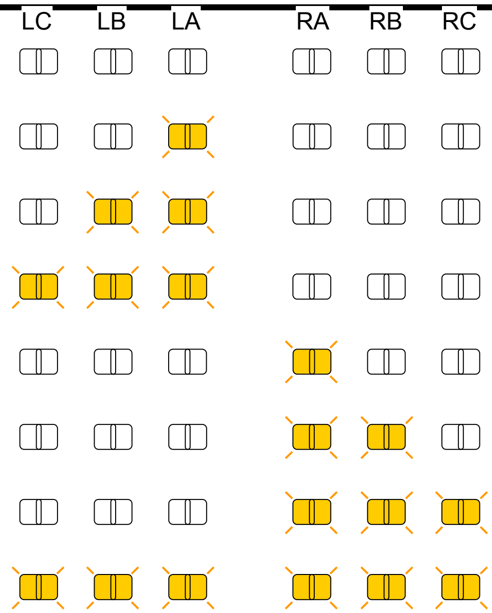
Identifying Inputs and Outputs

- Left-turn signal lever (LEFT)
- Right-turn signal lever (RIGHT)

- Hazard lights push-button (HAZ)
- <u>OUTPUTS</u>: LA, LB, C, RA, RB, RC

Identifying States

and their Symbolic Names



← IDLE

← L1

← L2

← L3

← R1

← R2

← R3

← LR3 (hazard lights)

Initial State Diagram

- Moore machine: output depends only on current state
- Label "1" on a transition arc indicates that all input is ignored and the machine transitions to the next state upon completion of current-state/output computation

Output Table

State	LC	LB	LA	RA	RB	RC
IDLE	0	0	0	0	0	0
L1	0	0	1	0	0	0
L2	0	1	1	0	0	0
L3	1	1	1	0	0	0
R1	0	0	0	1	0	0
R2	0	0	0	1	1	0
R3	0	0	0	1	1	1
LR3	1	1	1	1	1	1

← Hazard

Output Equations

LA = L1 + L2 + L3 + LR3

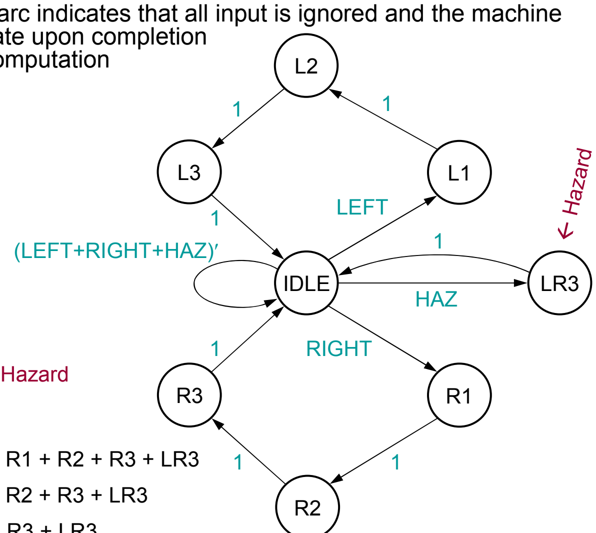
LB = L2 + L3 + LR3

LC = L3 + LR3

RA = R1 + R2 + R3 + LR3

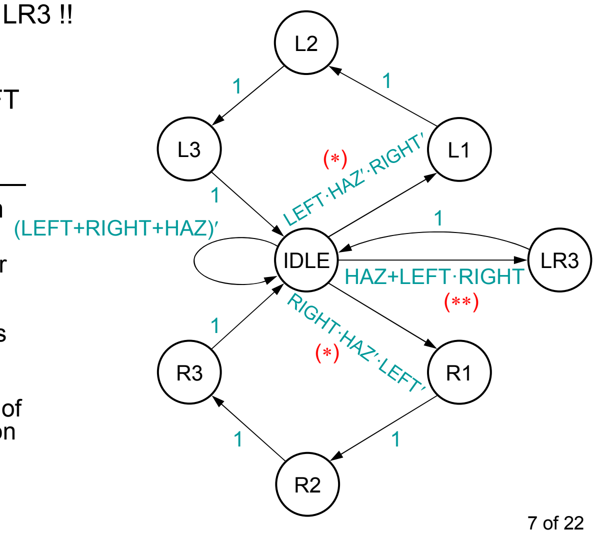
RB = R2 + R3 + LR3

RC = R3 + LR3



Corrected State Diagram

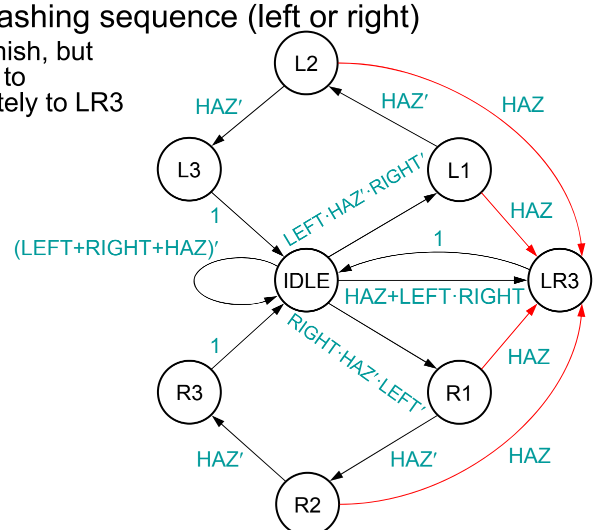
- **Problem**: doesn't properly handle multiple inputs asserted simultaneously e.g., if in IDLE both LEFT and HAZ are asserted the machine goes to 2 states: L1 and LR3 !!
- Corrections:
  (*) give HAZ priority;
  (**) treat concurrent LEFT and RIGHT as HAZ
- The corrected state diagram is unambiguous—transition expressions on arcs are:
  - **mutually exclusive**: for each state, the product of any pair of outgoing transition expressions is "0"
  - **all-inclusive**: for each state, the sum of transition expressions on all outgoing arcs is "1"



7 of 22

Enhanced State Diagram

- Problem: What if HAZ is activated while the machine cycles through a flashing sequence (left or right)
  - The cycle would finish, but a better solution is to transition immediately to LR3



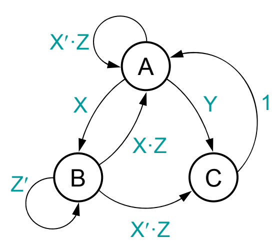
8 of 22

Ambiguity in State Diagram

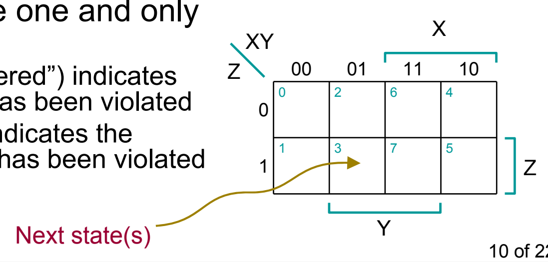
- In a properly constructed state diagram, each input combination is covered exactly once by an expression of an outgoing arc
- Ambiguous: double-covered or uncovered
- Some input combinations are covered by more than one expressions (double-covered)
  - Given such an input combination, there are two or more next-state-transitions to follow
  - **Mutual exclusion:** AND of any pair of expressions should be “0”
- Some input combinations are not covered by any expressions (uncovered)
  - Given such an input combination, there are no any next-state-transitions to follow
  - **All-inclusion:** OR of all expressions should be “1”

State Diagram Ambiguities - Example

- Example state diagram:
  - States: A, B, C
  - Inputs: X, Y, Z
- Karnaugh-like maps:
  - Consider each state separately
    - Draw a different K-map for each state
  - Each cell represents a unique combination of inputs
  - For all outgoing transitions fill in the corresponding cells with next state
- Each cell should have one and only one entry:
  - An *empty* cell (“uncovered”) indicates the All-inclusion rule has been violated
  - *More than one entry* indicates the Mutual-exclusion rule has been violated

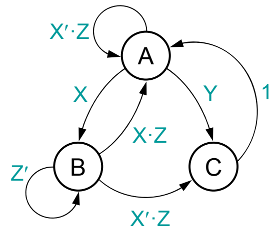


Current state:



State Diagram Ambiguities - Example

- **in state A**:
  - for input X=1, transition to next-state B
    ➔ fill in corresponding K-map cells
    with "**B**"
  - for input Y=1, transition to C
    ➔ fill in corresponding K-map cells
    with "**C**"
  - for input X=0,Z=1, transition to A
    ➔ fill in corresponding K-map cells
    with "**A**"



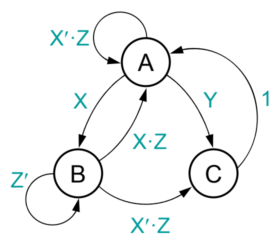
Current state **A**:

Z \ XY	00	01	11	10
0		C	B+C	B
1	A	A+C	B+C	B

- Empty cell ("uncovered"):
  - All-inclusion rule violated
- Multiple entries ("doubly covered"):
  - Mutual-exclusion rule violated
- State **A** not unambiguously specified !!

State Diagram Ambiguities - Example

- **in state B**:
  - for input Z=0, transition to next-state B
    ➔ fill in corresponding K-map cells
    with "**B**"
  - for input X=1,Z=1, transition to A
    ➔ fill in corresponding K-map cells
    with "**A**"
  - for input X=0,Z=1, transition to C
    ➔ fill in corresponding K-map cells
    with "**C**"



Current state **B**:

Z \ XY	00	01	11	10
0	B	B	B	B
1	C	C	A	A

- No ambiguities — each cell has one and only one entry
- State **B** is unambiguously specified !!
- NOTE: State **C** is unambiguously specified because of the unconditional transition to A (indicated by "1" on the outgoing arc)

State Assignment

(Back to the T-bird lights example…)

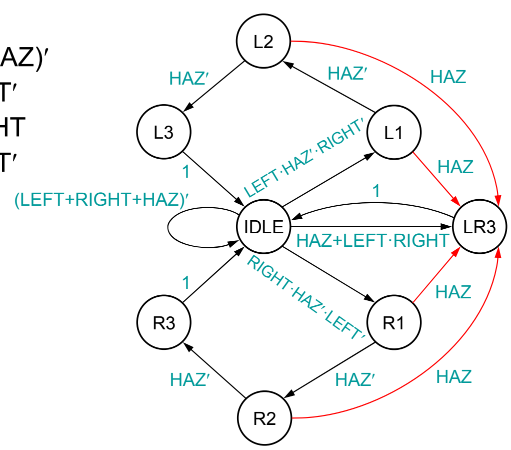
- For 8 states, need 3 flip-flops
- Initial (IDLE) state coded as "000" for easy reset
- State variable Q2 used to distinguish "left" vs. "right"
- State variables Q1 and Q0 used to "count" in Gray-code sequence: IDLE→L1→L2→L3→IDLE
  - Minimizes the number of state-variable changes per transition
- The remaining binary combination "100" used for the LR3 state

State Assignment

State	Q2	Q1	Q0
IDLE	0	0	0
L1	0	0	1
L2	0	1	1
L3	0	1	0
R1	1	0	1
R2	1	1	1
R3	1	1	0
LR3	1	0	0

Listing Next State Transitions

- From state IDLE:
  - Input: (LEFT+RIGHT+ HAZ)′
  - Input: LEFT·HAZ′·RIGHT′
  - Input: HAZ + LEFT·RIGHT
  - Input: RIGHT·HAZ′·LEFT′
- From state L1:
  - Input: HAZ
  - Input: HAZ′
- From state L2:
  - Input: HAZ
  - Input: HAZ′
- From state L3:
  - Input: 1 (state completion transition)
- …



Transition List

- Similar to a *transition table*, but transitions in the state diagram are specified by expressions, not by an extensive tabulation of next states

Current State | Next State

Outgoing Transitions

S	Q2	Q1	Q0	Transition Expression	S∗	Q2∗	Q1∗	Q0∗
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

From a transition list, circuit synthesis is just "turning-the-crank," — automated using a CAD tool

Synthesizing Circuit from Transition List

- Transition equation

  $V* = \sum_{\text{transition-list rows where } V*=1}$ (transition p-term)

  – A p-term is the product of current state's minterm and the transition expression
- The transition equation for Q2∗ is the T-bird machine:

  Q2∗ = Q2′·Q1′·Q0′ · (HAZ + LEFT·RIGHT)
  + Q2′·Q1′·Q0′ · (RIGHT·HAZ′·LEFT′)
  + Q2′·Q1′·Q0 · (HAZ)
  + Q2′·Q1·Q0 · (HAZ)
  + Q2·Q1′·Q0 · (HAZ′)
  + Q2·Q1′·Q0 · (HAZ)
  + Q2·Q1·Q0 · (HAZ′)
  + Q2·Q1·Q0 · (HAZ)

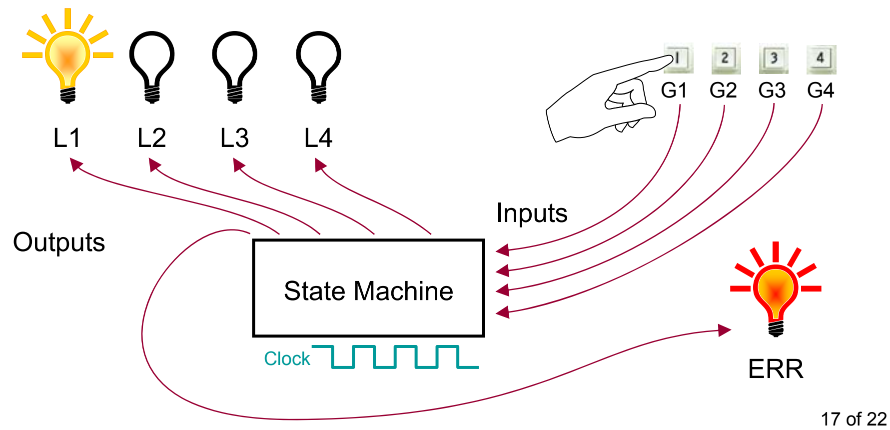
S	Q2	Q1	Q0	Transition Expression	S∗	Q2∗	Q1∗	Q0∗
IDLE	0	0	0	(LEFT + RIGHT + HAZ)′	IDLE	0	0	0
IDLE	0	0	0	LEFT · HAZ′ · RIGHT′	L1	0	0	1
IDLE	0	0	0	HAZ + LEFT · RIGHT	LR3	1	0	0
IDLE	0	0	0	RIGHT · HAZ′ · LEFT′	R1	1	0	1
L1	0	0	1	HAZ′	L2	0	1	1
L1	0	0	1	HAZ	LR3	1	0	0
L2	0	1	1	HAZ′	L3	0	1	0
L2	0	1	1	HAZ	LR3	1	0	0
L3	0	1	0	1	IDLE	0	0	0
R1	1	0	1	HAZ′	R2	1	1	1
R1	1	0	1	HAZ	LR3	1	0	0
R2	1	1	1	HAZ′	R3	1	1	0
R2	1	1	1	HAZ	LR3	1	0	0
R3	1	1	0	1	IDLE	0	0	0
LR3	1	0	0	1	IDLE	0	0	0

- After simplification:

  Q2∗ = Q2′·Q1′·Q0′·(HAZ+RIGHT) + Q2′·Q0·(HAZ) + Q2·Q0
  Q1∗ = Q0·HAZ′
  Q0∗ = Q2′·Q1′·Q0′·HAZ′·(LEFT⊕RIGHT) + Q1′·Q0·HAZ′
- Note: These equations are not necessarily minimal

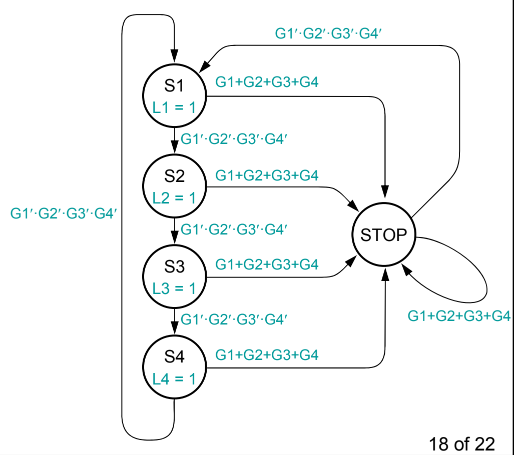
Another State-Machine Design Example

- The Guessing Game
  - 1-out-of-4 lamps lit
  - At each clock tick, the pattern is rotated by one
  - Make a guess by pressing a button Gi:
    - If Gi = asserted(Li) play stops
    - If Gi ≠ asserted(Li) play stops and ERR lamp is lit



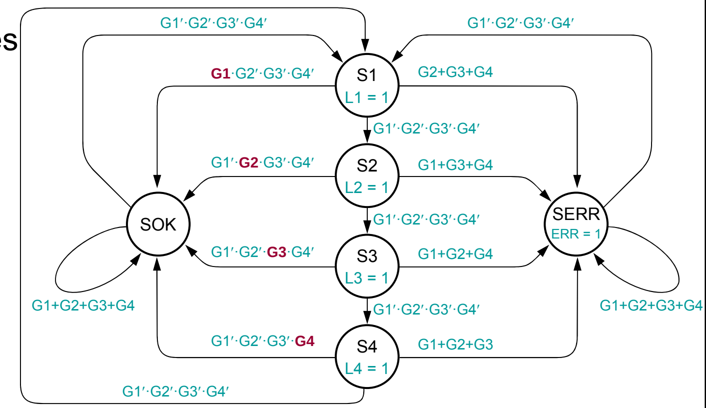
State Diagram - First Try

- Machine cycles through states S1–S4 as long as no Gi is asserted
- Goes to STOP when a guess is made
- PROBLEM: In STOP, doesn't "remember" if guess was correct, so cannot control ERR lamp
  - Moore machine: output depends on current state only



State Diagram – Corrected

- Solution: two “stopped” states, SOK and SERR
  - SOK = correct guess
  - SERR = wrong guess → assert ERR output
- Machine goes to SERR if user presses ≥2 buttons at once or changes guess while in STOP



Transition List for Guessing Game

Outgoing Transitions

Current State					Next State				Output				
S	Q2	Q1	Q0	Transition Expression	S*	Q2*	Q1*	Q0*	L1	L2	L3	L4	ERR
S1	0	0	0	G1′·G2′·G3′·G4′	S2	0	0	1	1	0	0	0	0
S1	0	0	0	G1·G2′·G3′·G4′	SOK	1	0	0	1	0	0	0	0
S1	0	0	0	G2+G3+G4	SERR	1	0	1	1	0	0	0	0
S2	0	0	1	G1′·G2′·G3′·G4′	S3	0	1	1	0	1	0	0	0
S2	0	0	1	G1′·G2·G3′·G4′	SOK	1	0	0	0	1	0	0	0
S2	0	0	1	G1+G3+G4	SERR	1	0	1	0	1	0	0	0
S3	0	1	1	G1′·G2′·G3′·G4′	S4	0	1	0	0	0	1	0	0
S3	0	1	1	G1′·G2′·G3·G4′	SOK	1	0	0	0	0	1	0	0
S3	0	1	1	G1+G2+G4	SERR	1	0	1	0	0	1	0	0
S4	0	1	0	G1′·G2′·G3′·G4′	S1	0	0	0	0	0	0	1	0
S4	0	1	0	G1′·G2′·G3′·G4	SOK	1	0	0	0	0	0	1	0
S4	0	1	0	G1+G2+G3	SERR	1	0	1	0	0	0	1	0
SOK	1	0	0	G1+G2+G3+G4	SOK	1	0	0	0	0	0	0	0
SOK	1	0	0	G1′·G2′·G3′·G4′	S1	0	0	0	0	0	0	0	0
SERR	1	0	1	G1+G2+G3+G4	SERR	1	0	1	0	0	0	0	1
SERR	1	0	1	G1′·G2′·G3′·G4′	S1	0	0	0	0	0	0	0	1

Transition Equations

- Obtained from the transition list—**transition equations**:

$Q1* = Q2' \cdot Q0 \cdot G1' \cdot G2' \cdot G3' \cdot G4'$

$Q0* = Q2' \cdot Q1' \cdot Q0' \cdot (G1' \cdot G2' \cdot G3' \cdot G4')$
$+ Q2' \cdot Q1' \cdot Q0' \cdot (G2+G3+G4)$
$+ Q2' \cdot Q1' \cdot Q0 \cdot (G1' \cdot G2' \cdot G3' \cdot G4')$
$+ Q2' \cdot Q1' \cdot Q0 \cdot (G1+G3+G4)$
$+ Q2' \cdot Q1 \cdot Q0 \cdot (G1+G2+G4)$
$+ Q2' \cdot Q1 \cdot Q0' \cdot (G1+G2+G3)$
$+ Q2 \cdot Q1' \cdot Q0 \cdot (G1+G2+G3+G4)$

$Q2*' = (Q2'+Q1') \cdot (G1' \cdot G2' \cdot G3' \cdot G4')$ ← formulated for "0"s

- Moore machine → outputs independent of transition expressions → only one row of transition list must be considered for each current state
- **Output equations**:

$L1 = Q2' \cdot Q1' \cdot Q0'$ $L3 = Q2' \cdot Q1 \cdot Q0$ $ERR = Q2 \cdot Q1' \cdot Q0$
$L2 = Q2' \cdot Q1' \cdot Q0$ $L4 = Q2' \cdot Q1 \cdot Q0'$

21 of 22

Unused States

- Guessing machine state diagram has 6 states, but 3 flip-flops have 8 states
- Omitted unused states implicitly treated as "don't-cares" in next state equations:
  - Equations for Q1∗ and Q0∗ written as sum of transition p-terms for state/input combinations with explicit "1" in Q1∗ and Q0∗ columns
  - Unused states implicitly assumed to have "0" in Q1∗ and Q0∗ columns
  - Equation for Q2∗′ written as a sum of transition p-terms for state/input combinations with explicit "0" in Q2∗ column
- As a consequence, all unused states have a coded next state of "100" for all input combinations
  == coding for SOK state
- This is safe & acceptable, but could treat them explicitly as "don't-cares" – see Wakerly, 4^{th} ed., page 583

22 of 22