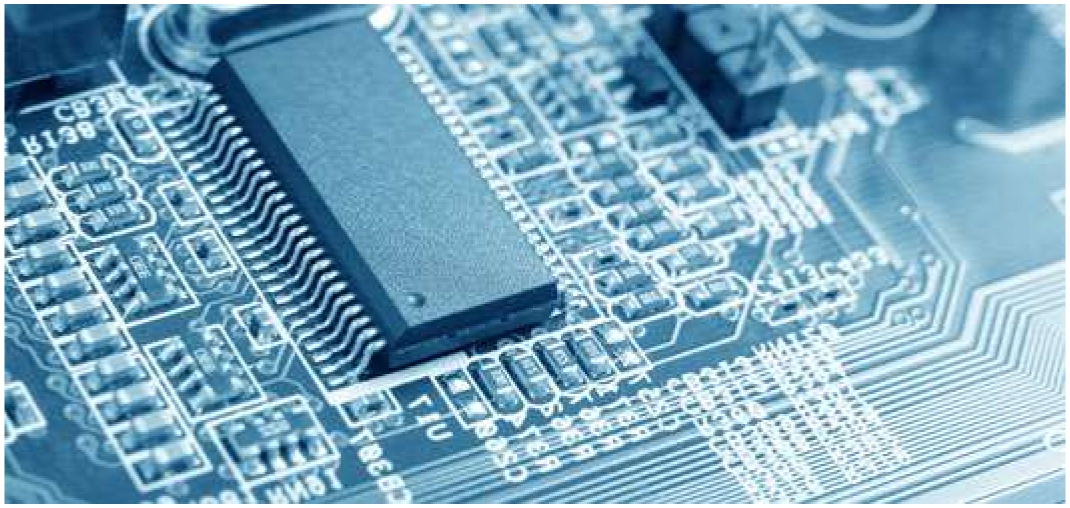
NAZARBAYEV UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY



ROBT206 – Microcontrollers with Lab

Lecture 11 – Combinational Logic Design Continued

15 February, 2018

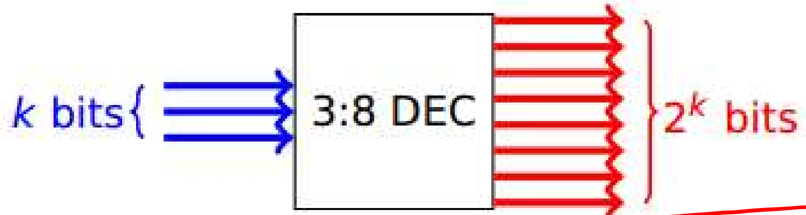
Topics

Today's Topics
Encoders
Selecting using Multiplexers

Encoding

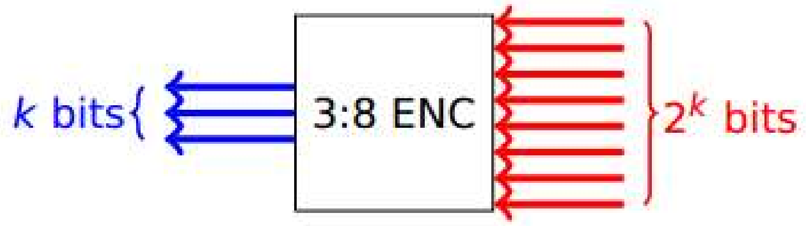
- Encoding - the opposite of decoding: the conversion of an *m*-bit input code to a *n*-bit output code with $n \leq m \leq 2^n$ such that each valid code word produces a unique output code
- Circuits that perform encoding are called *encoders*
- An encoder has 2^n (or fewer) input lines and *n* output lines which generate the binary code corresponding to the input values
- Typically, an encoder converts a code containing exactly one bit that is 1 to a binary code corresponding to the position in which the 1 appears.

Encoders and Decoders



$A_0 = D_7 + D_5 + D_3 + D_1$

BCD			One-Hot D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	$\boxed{1}$	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	$\boxed{1}$	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	$\boxed{1}$	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	$\boxed{1}$	0	0	0	0	0	0	0



Encoder Example

$\boxed{A_3 = D_9 + D_8}$

- A decimal-to-BCD encoder
  - Inputs: 10 bits corresponding to decimal digits 0 through 9, (D_0, …, D_9)
  - Outputs: 4 bits with BCD codes
  - Function: If input bit D_i is 1, then the output (A_3,A_2,A_1,A_0) is the BCD code for *i*,
- The truth table could be formed, but alternatively, the equations for each of the four outputs can be obtained directly.

	D_9	D_8	D_7	. . .	D_4	D_3	D_2 -	D_1	D_0	A_3	A_2	A_1	A_0
input = 0	0				0	0	0	0	1	0	0	0	0
1								1	0	0	0	0	1
2							1	0	0	0	0	1	0
3						1				0			
⋮										0	⋮		
8		1				1				1	0	0	0
9	1									1	0	0	1

Encoder Example (continued)

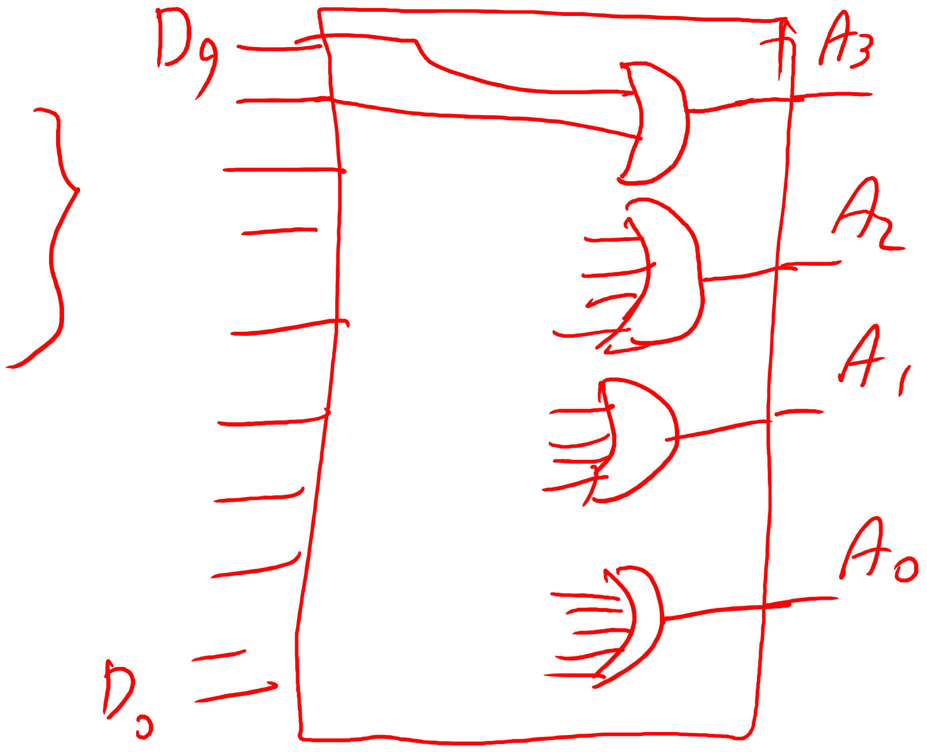
- Input D_i is a term in equation A_j if bit A_j is 1 in the binary value for i.
- Equations:

$A_3 = D_8 + D_9$ ✓

$A_2 = D_4 + D_5 + D_6 + D_7$ ✓

$A_1 = D_2 + D_3 + D_6 + D_7$

$A_0 = D_1 + D_3 + D_5 + D_7 + D_9$



Priority Encoder

- If more than one input value is 1, then the encoder just designed does not work.
- One encoder that can accept all possible combinations of input values and produce a meaningful result is a *priority encoder*.
- Among the 1s that appear, it selects the most significant input position (or the least significant input position) containing a 1 and responds with the corresponding binary code for that position.

Priority Encoder Example

- Priority encoder with 5 inputs (D_4, D_3, D_2, D_1, D_0) - highest priority to most significant 1 present - Code outputs A2, A1, A0 and V where V indicates at least one 1 present.

No. of Min-terms/Row	Inputs : 5 inputs					Outputs			
	D4	D3	D2	D1	D0	A2	A1	A0	V
1	0	0	0	0	0	X	X	X	0
1	0	0	0	0	1	0	0	0	1
2	0	0	0	1	X	0	0	1	1
4	0	0	1	X	X	0	1	0	1
8	0	1	X	X	X	0	1	1	1
16	1	X	X	X	X	1	0	0	1

The input is not valid

- Xs in input part of table represent 0 or 1; thus table entries correspond to product terms instead of minterms. The column on the left shows that all 32 minterms are present in the product terms in the table

Priority Encoder Example (continued)

- Could use a K-map to get equations, but can be read directly from table and manually optimized if careful:

$$A_2 = D_4$$

$$A_1 = \overline{D_4}\,D_3 + \overline{D_4}\,\overline{D_3}\,D_2 = \overline{D_4}\,F_1,\quad F_1 = (D_3 + D_2)$$

$$A_0 = \overline{D_4}\,D_3 + \overline{D_4}\,\overline{D_3}\,\overline{D_2}\,D_1 = \overline{D_4}\,(D_3 + \overline{D_2}\,D_1)$$

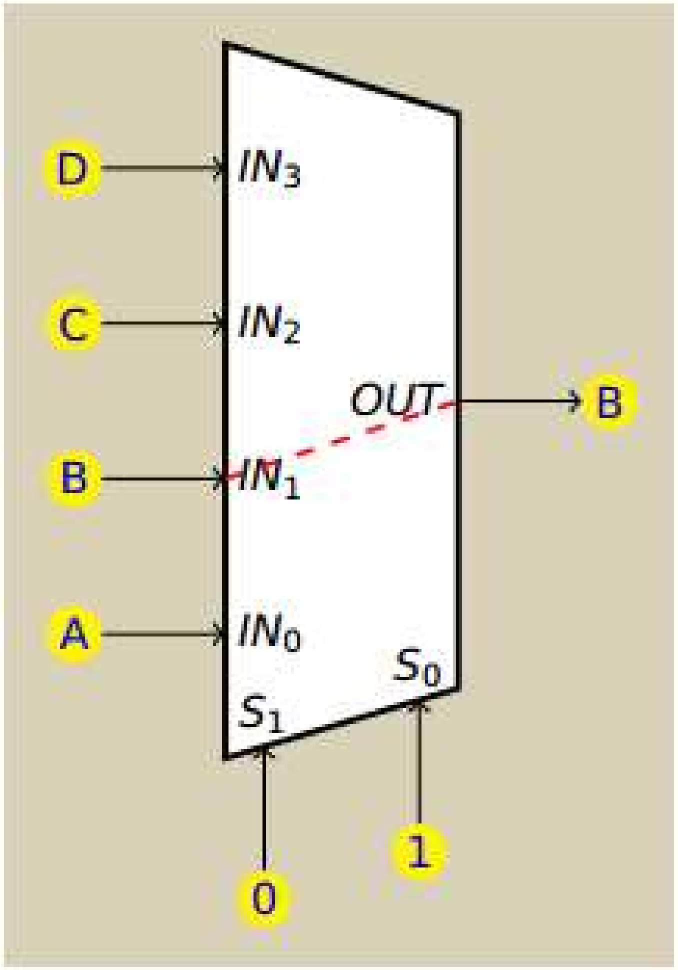
$$V = D_4 + F_1 + D_1 + D_0$$

Selecting

- Selecting of data or information is a critical function in digital systems and computers
- Circuits that perform selecting have:
  - A set of information inputs from which the selection is made
  - A single output
  - A set of control lines for making the selection
- Logic circuits that perform selecting are called ***multiplexers***

Multiplexers

- A multiplexer selects information from an input line and directs the information to an output line
- A typical multiplexer has *n* control inputs ($S_{n-1}, \ldots S_0$) called *selection inputs*, 2^n information inputs ($I_{2^n-1}, \ldots I_0$), and one output Y
- A multiplexer can be designed to have *m* information inputs with $m < 2^n$ as well as *n* selection inputs

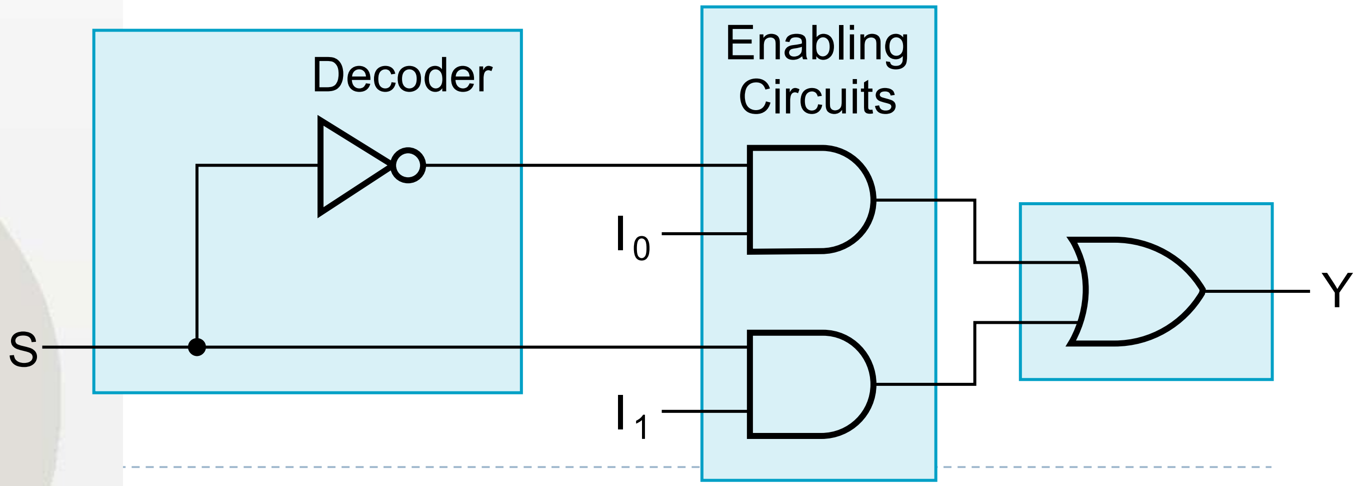


2-to-1-Line Multiplexer

- Since $2 = 2^1$, $n = 1$
- The single selection variable S has two values:
  - $S = 0$ selects input I_0
  - $S = 1$ selects input I_1
- The equation:

$$Y = \overline{S} I_0 + S I_1$$

- The circuit:

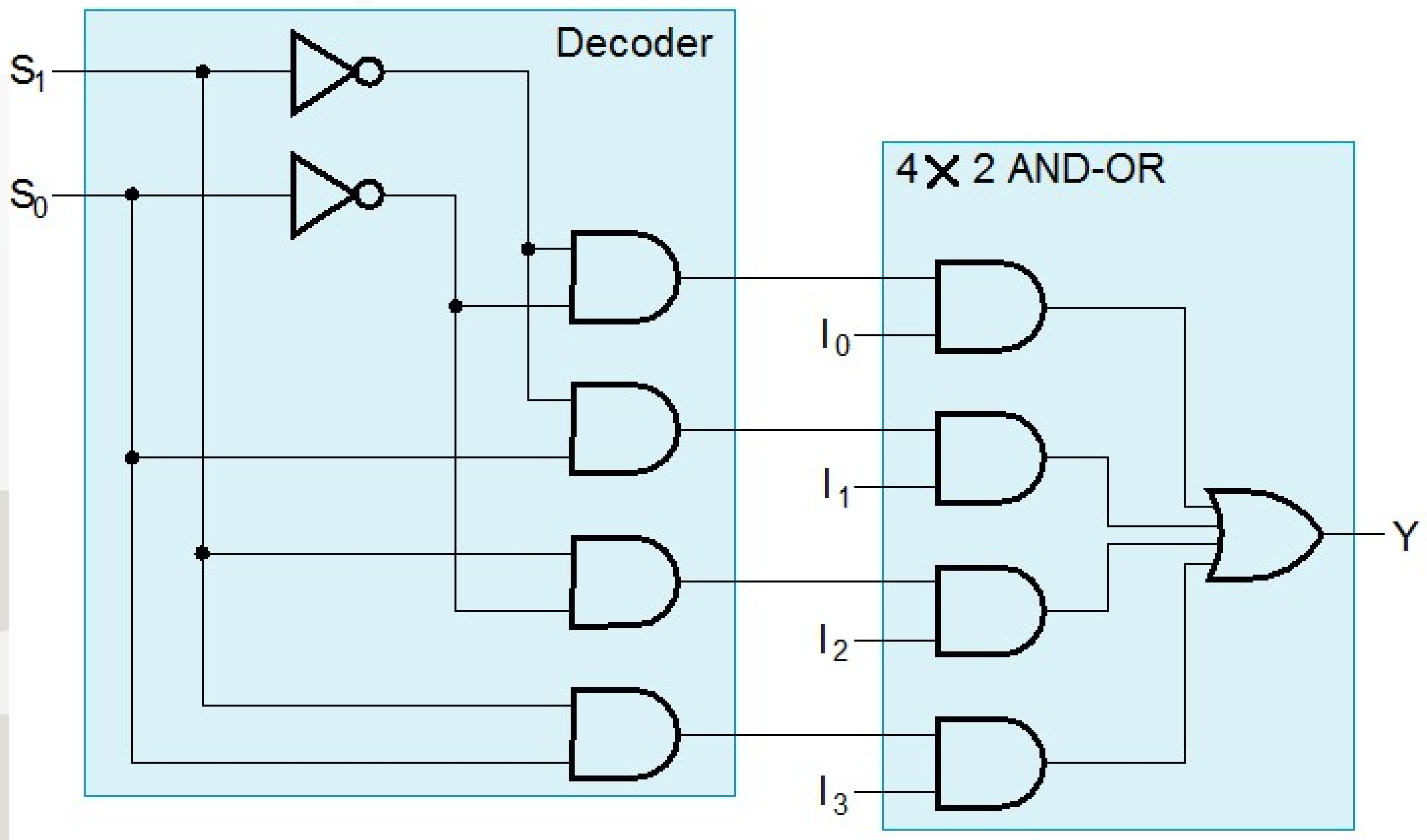


2-to-1-Line Multiplexer (continued)

- Note the regions of the multiplexer circuit shown:
  - 1-to-2-line Decoder
  - 2 Enabling circuits
  - 2-input OR gate
- To obtain a basis for multiplexer expansion, we combine the Enabling circuits and OR gate into a 2×2 AND-OR circuit:
  - 1-to-2-line decoder
  - 2×2 AND-OR
- In general, for an 2^n-to-1-line multiplexer:
  - n-to-2^n-line decoder
  - $2^n \times 2$ AND-OR

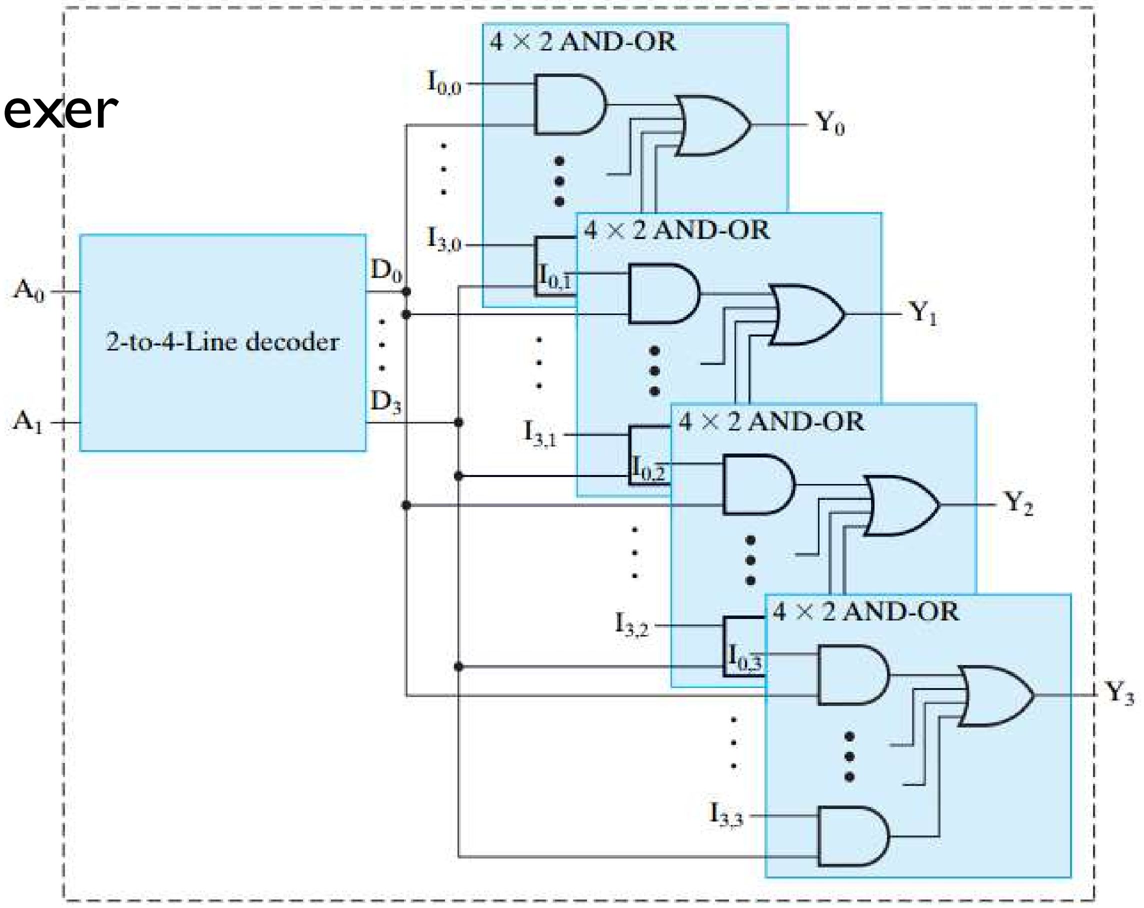
Example: 4-to-1-line Multiplexer

- 2-to-2^2-line decoder
- $2^2 \times$ 2 AND-OR

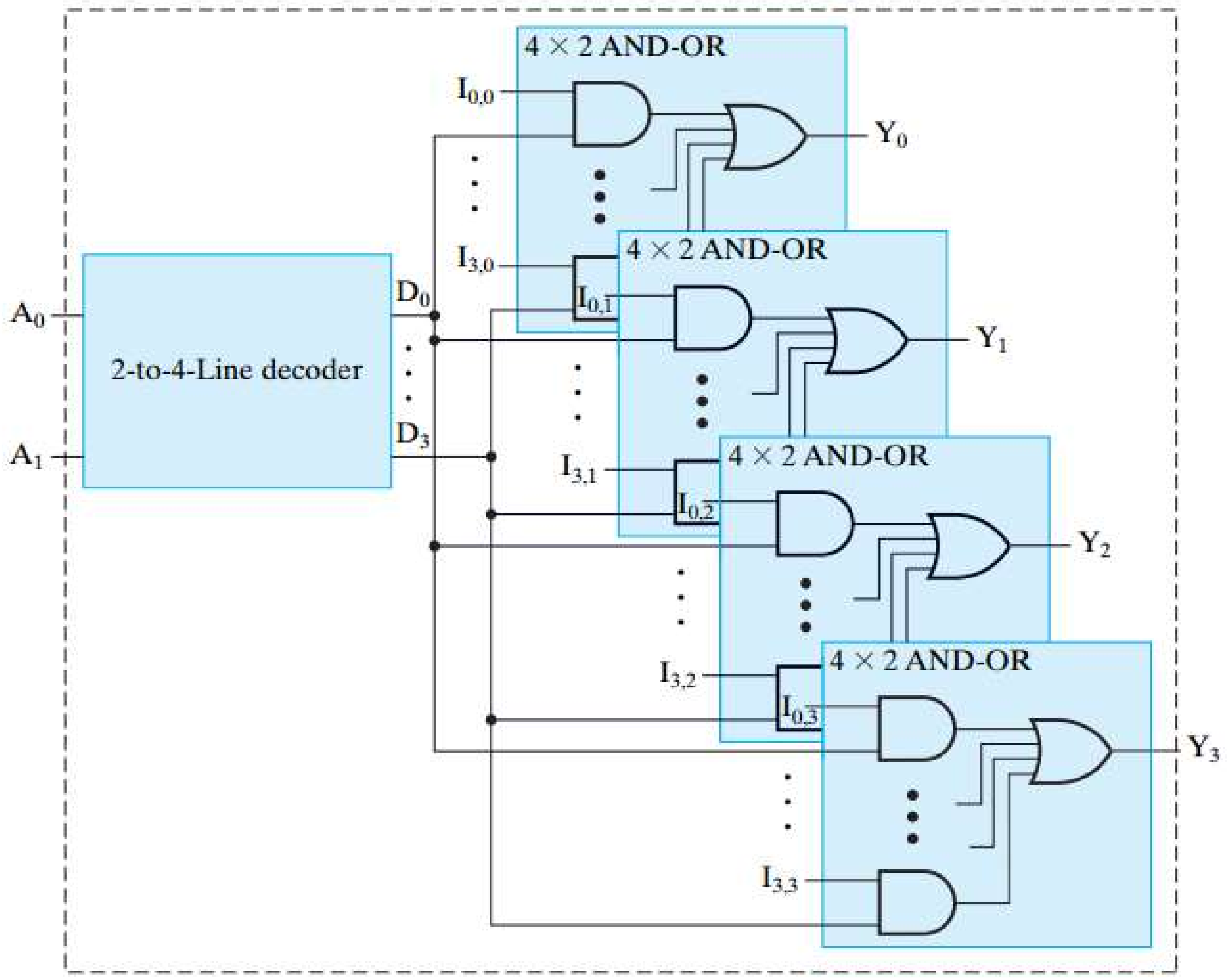


Multiplexer Width Expansion

- Select "vectors of bits" instead of "bits"
- Use multiple copies of $2^n \times 2$ AND-OR in parallel
- Example:
  4-to-1-line quad multiplexer



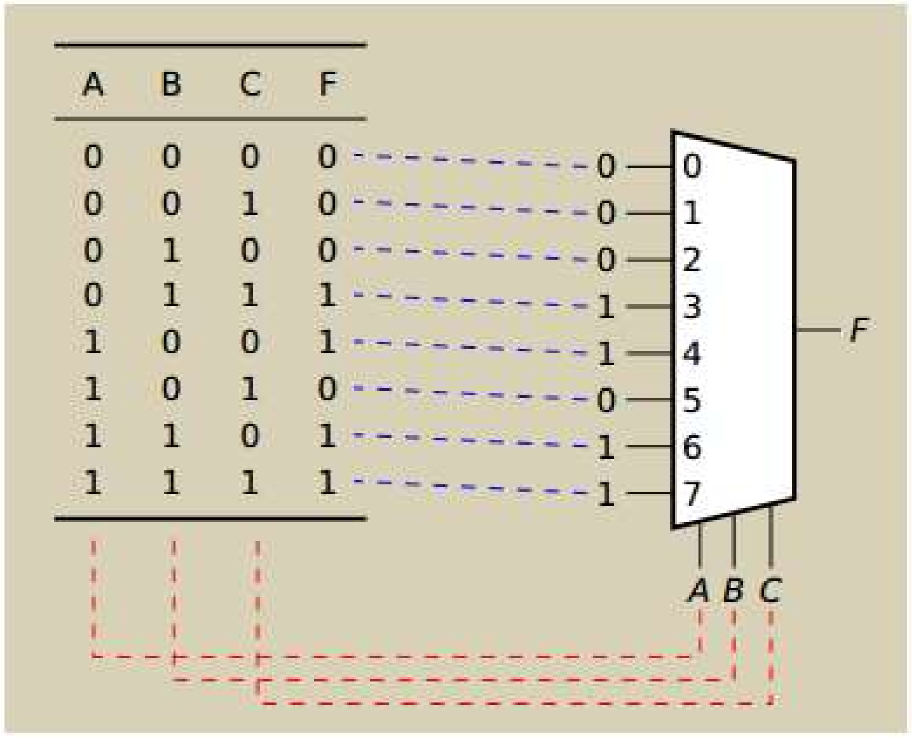
Multiplexer Width Expansion



Combinational Logic Implementation Approach 1

Think of a function as using *k* input bits to choose from 2^k outputs.

E.g., $F = BC + A\overline{C}$



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Combinational Logic Implementation - Multiplexer Approach

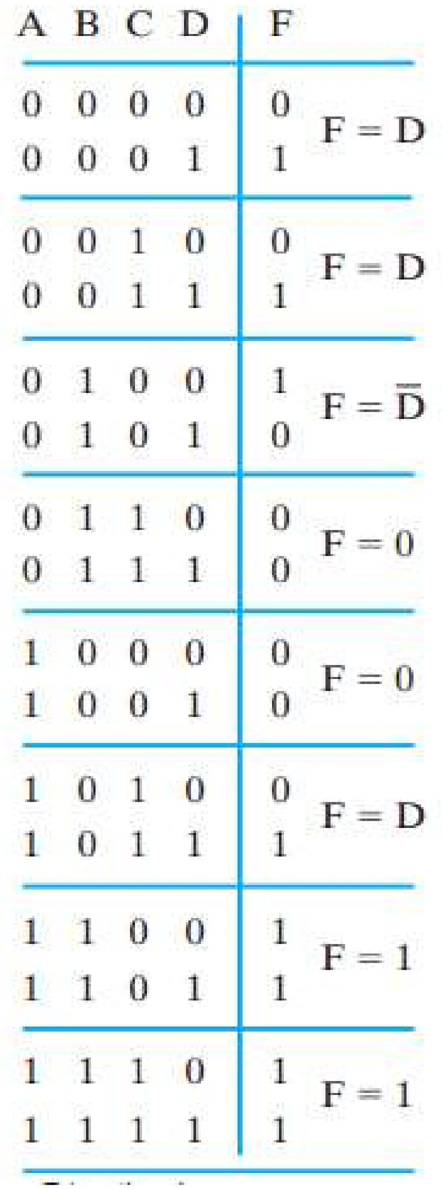
- Implement any *m* functions of *n* + 1 variables by using:
  - An m-wide 2^n-to-1-line multiplexer
  - A single inverter
- Design:
  - Find the truth table for the functions.
  - Based on the values of the first *n* variables, separate the truth table rows into pairs
  - For each pair and output, define a rudimentary function of the final variable (0, 1, X, $\overline{X}$)
  - Using the first *n* variables as the index, value-fix the information inputs to the multiplexer with the corresponding rudimentary functions
  - Use the inverter to generate the rudimentary function $\overline{X}$

Example: 4 Variable Function

- Implement the Boolean function

F(A, B, C, D) = $\Sigma m(1,3,4,11,12,13,14,15)$

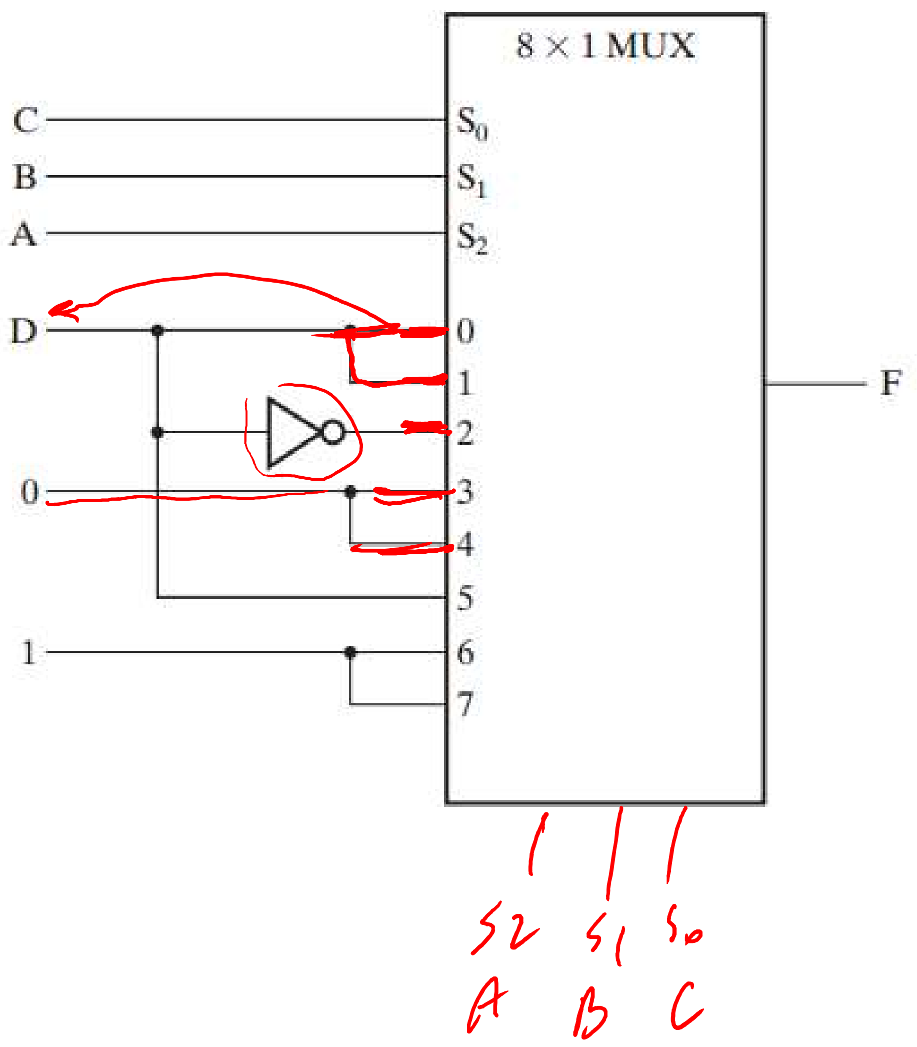
- Variables A, B, C are connected to selection inputs S_2, S_1, S_0 , respectively
- Values for the data inputs are defined from the truth table.
- Example: if (A,B,C) = 001, the truth table shows F = D, variable D is applied to I_1



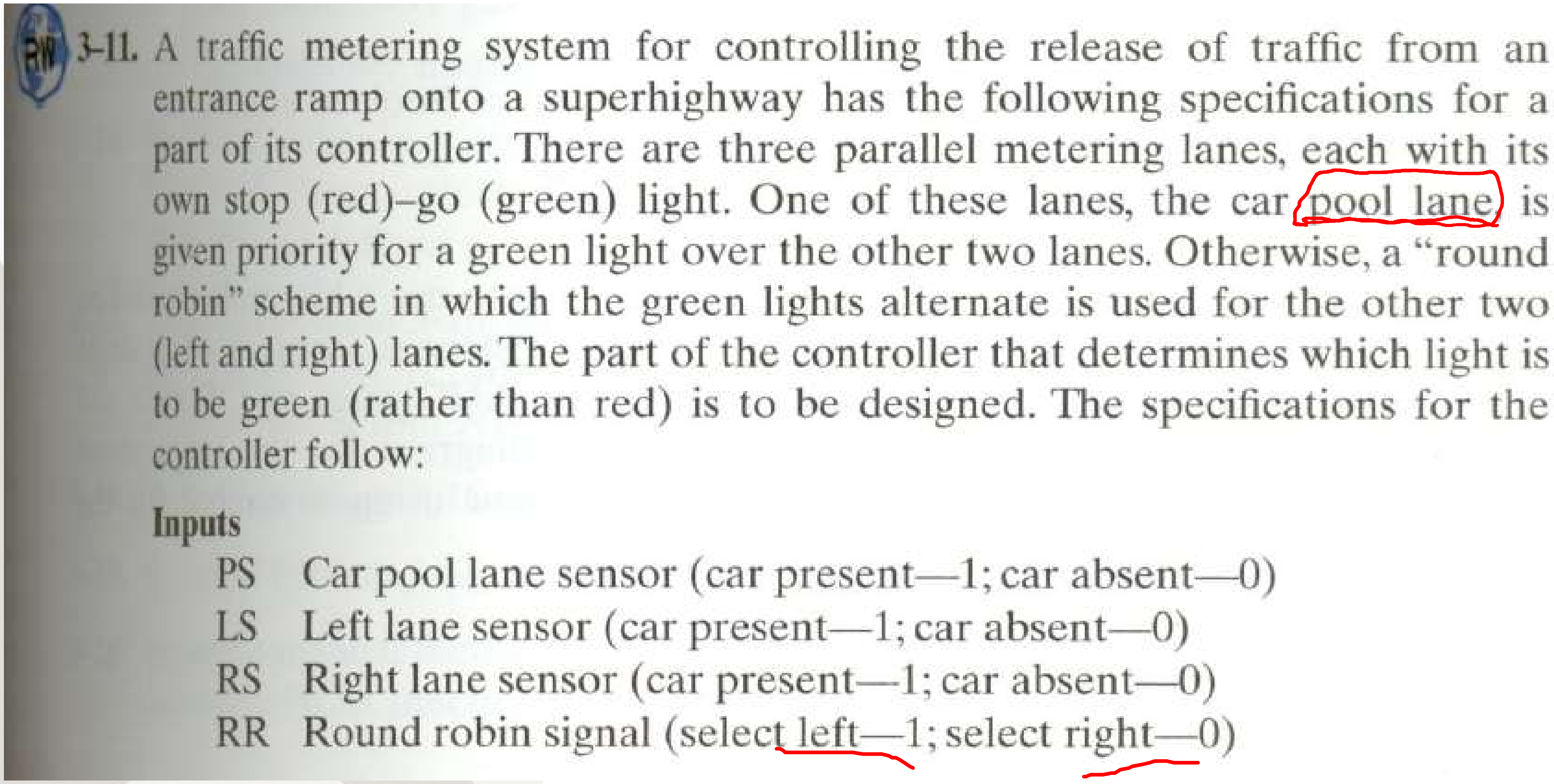
A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = \bar{D}$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

Example: 4 Variable Function

A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



Example 3.11



3-11. A traffic metering system for controlling the release of traffic from an entrance ramp onto a superhighway has the following specifications for a part of its controller. There are three parallel metering lanes, each with its own stop (red)–go (green) light. One of these lanes, the car pool lane, is given priority for a green light over the other two lanes. Otherwise, a "round robin" scheme in which the green lights alternate is used for the other two (left and right) lanes. The part of the controller that determines which light is to be green (rather than red) is to be designed. The specifications for the controller follow:

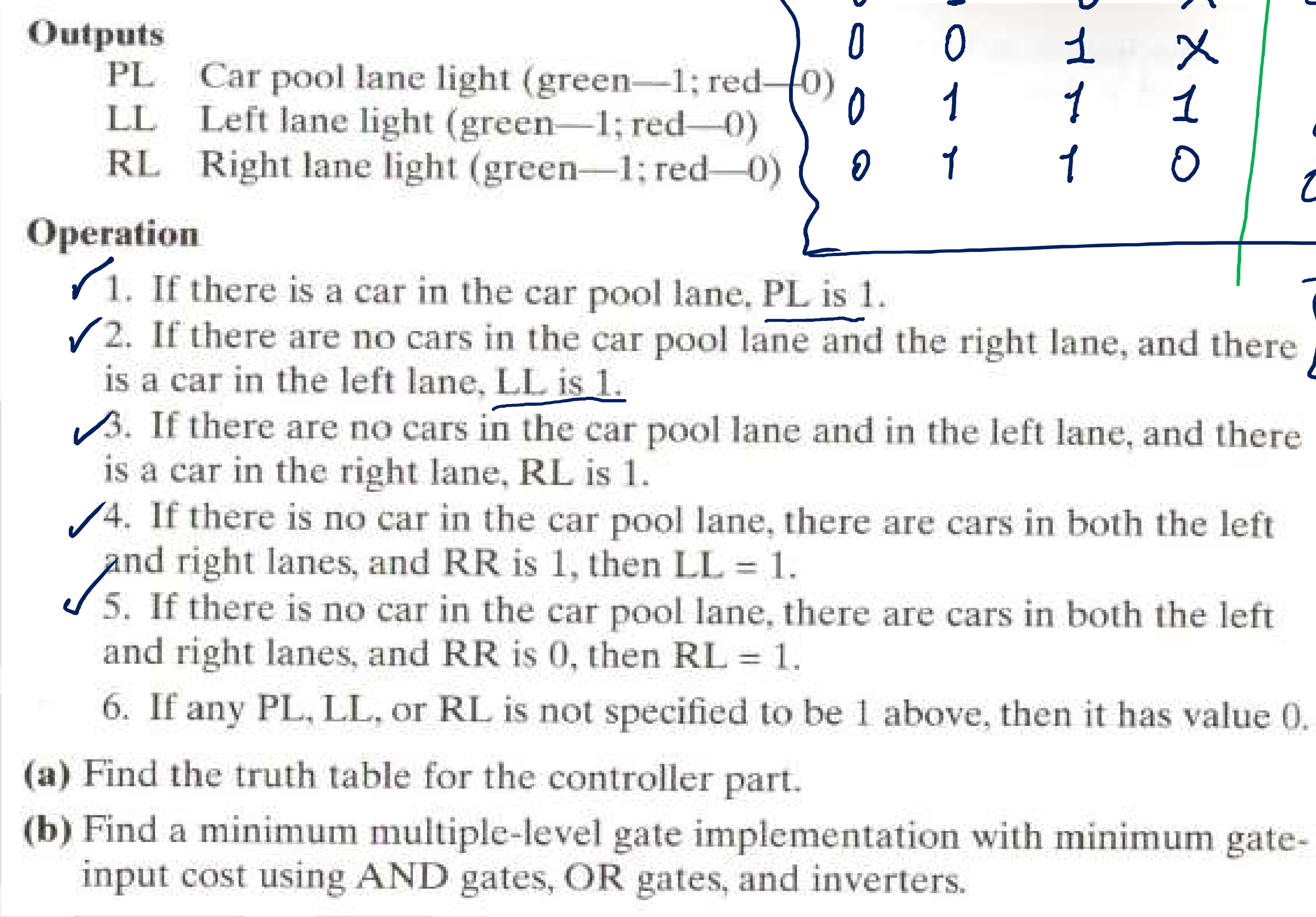
Inputs

PS	Car pool lane sensor (car present—1; car absent—0)
LS	Left lane sensor (car present—1; car absent—0)
RS	Right lane sensor (car present—1; car absent—0)
RR	Round robin signal (select left—1; select right—0)

Example 3.11

PS	LS	RS	RR	PL	LL	RL
1	x	x	X	1	0	0
0	1	0	X	0	1	0
0	0	1	X	0	0	1
0	1	1	1	0	1	0
0	1	1	0	0	0	1

$\boxed{PL \overset{?}{=} PS}$



Outputs

PL	Car pool lane light (green—1; red—0)
LL	Left lane light (green—1; red—0)
RL	Right lane light (green—1; red—0)

Operation

✓1. If there is a car in the car pool lane, PL is 1.
✓2. If there are no cars in the car pool lane and the right lane, and there is a car in the left lane, LL is 1.
✓3. If there are no cars in the car pool lane and in the left lane, and there is a car in the right lane, RL is 1.
✓4. If there is no car in the car pool lane, there are cars in both the left and right lanes, and RR is 1, then LL = 1.
✓5. If there is no car in the car pool lane, there are cars in both the left and right lanes, and RR is 0, then RL = 1.
6. If any PL, LL, or RL is not specified to be 1 above, then it has value 0.

(a) Find the truth table for the controller part.

(b) Find a minimum multiple-level gate implementation with minimum gate-input cost using AND gates, OR gates, and inverters.

Any Questions?