## Ch 5. Combinational Logic Case Studies

## Combinational logic design case studies

- General design procedure
- Case studies
  - process line controller
  - telephone keypad decoder
  - calendar subsystem

# General design procedure for combinational logic

- 1. Understand the problem
  - what is the circuit supposed to do?
  - write down inputs (data, control) and outputs
  - draw block diagram or other picture
- 2. Formulate the problem using a suitable design representation
  - truth table or waveform diagram are typical
  - may require encoding of symbolic inputs and outputs
- 3. Choose implementation target
  - ROM, PAL, PLA
  - mux, decoder and OR-gate
  - discrete gates
- 4. Follow implementation procedure
  - K-maps for two-level, multi-level
  - design tools and hardware description language (e.g., Verilog)

#### Production line control

- Rods of varying length (+/-10%) travel on conveyor belt
  - mechanical arm pushes rods within spec (+/-5%) to one side
  - second arm pushes rods too long to other side
  - rods that are too short stay on belt
  - 3 light barriers (light source + photocell) as sensors
  - design combinational logic to activate the arms
- Understanding the problem
  - inputs are three sensors
  - outputs are two arm control signals
  - assume sensor reads "1" when tripped, "0" otherwise
  - call sensors A, B, C

#### Sketch of problem

#### Position of sensors

- A to B distance = specification -5%
- A to C distance = specification + 5%



#### Formalize the problem

#### Truth table

show	don't	cares

А	В	С	Function
0	0	0	do nothing
0	0	1	do nothing
0	1	0	do nothing
0	1	1	do nothing
1	0	0	too short
1	0	1	don't care
1	1	0	in spec
1	1	1	too long

logic implementation now straightforward just use three 3-input AND gates

"too short" = AB'C' (only first sensor tripped)

"in spec" = A B C' (first two sensors tripped)

"too long" = A B C (all three sensors tripped)

### Telephone Keypad Decoder

Design a combinational circuit that decodes a button press on a telephone keypad that has four rows and three columns of buttons



Decoded into a 4-bit binary number.

\* and # decoded into 10 and 15

## Telephone Keypad Decoder

•	<ul> <li>Step 1. Understand the problem</li> <li>Determine Input and Output Input : Four rows and three columns</li> </ul>	Module keypaddecoder(R1, R2, R3, R4, C1, 0 , C3, K8, K4, K2, K1, KP); Input R1, R2, R3, R4, C1, C2, C3; Output K8, K4, K2, K1, KP; Reg[3:0] key;				
	<ul> <li>Output : 4-bit binary number</li> <li>Assumptions <ul> <li>A button is pressed, corresponding row and col signals go to logic 1.</li> <li>Multiple press case is ignored.</li> </ul> </li> <li>Step 2. Formulate in a standard representation.</li> <li>Truth table is very large!(128 rows)</li> <li>Hardware description language</li> </ul>	Always @(R1, R2, R3, R4, C1, C2, C3) begin If R1 & C1 key =1; If R1 & C2 key =2; If R1 & C3 key =3; If R2 & C1 key =4; If R2 & C2 key =5; If R2 & C2 key =5; If R3 & C1 key =7; If R3 & C2 key =8; If R3 & C2 key =8; If R4 & C1 key =10; If R4 & C2 key =0; If R4 & C2 key =0; If R4 & C3 key =15; KP = ((R1+R2+R3+R4) == 3b`001)				
		Endmodule				

## Telephone Keypad Decoder

#### Step3. Implementation target

Transform the Verilog into logic equations.

 $K_1 = R_1C_1 + R_1C_3 + R_2C_2 + R_3C_1 + R_3C_3 + R_4C_3$ 

$$K_2 = R_1C_2 + R_1C_3 + R_2C_3 + R_3C_1 + R_4C_1 + R_4C_3$$

$$K_4 = R_2C_1 + R_2C_2 + R_2C_3 + R_3C_1 + R_4C_3$$

 $K_8 = R_3C_2 + R_3C_3 + R_4C_1 + R_4C_3$ 

KP has too many terms (12), so we implement KP`.

 $KP^{`} = R_1R_2 + R_1R_3 + R_1R_4 + R_2R_3 + R_2R_4 + R_3R_4 + C_1C_2 + C_1C_3 + C_2C_3$ 

- Step4. Implementation procedure.
  - In FPGA, we can decompose the equations by using intermediate terms

#### Calendar subsystem

Determine number of days in a month (to control watch display)

- used in controlling the display of a wrist-watch LCD screen
- inputs: month, leap year flag
- outputs: number of days
- Use software implementation to help understand the problem

```
integer number_of_days ( month, leap_year_flag) {
     switch (month) {
           case 1: return (31);
           case 2: if (leap_year_flag == 1)
                   then return (29)
                   else return (28);
           case 3: return (31);
           case 4: return (30);
           case 5: return (31);
           case 6: return (30);
           case 7: return (31);
           case 8: return (31);
           case 9: return (30);
           case 10: return (31);
           case 11: return (30);
           case 12: return (31);
           default: return (0);
```

#### Formalize the problem

#### Encoding:

- binary number for month: 4 bits
- 4 wires for 28, 29, 30, and 31
   one-hot only one true at any time
- Block diagram:



month	leap	28	29	30	31
0000		_	_	_	_
0001	_	0	0	0	1
0010	0	1	0	0	0
0010	1	0	1	0	0
0011	_	0	0	0	1
0100	_	0	0	1	0
0101	_	0	0	0	1
0110	_	0	0	1	0
0111	_	0	0	0	1
1000	_	0	0	0	1
1001	_	0	0	1	0
1010	_	0	0	0	1
1011	_	0	0	1	0
1100	_	0	0	0	1
1101	_	_	_	_	_
111–	_	_	_	_	_

## Choose implementation target and perform mapping

Discrete gates

	0000	_	_	_	_	_	
	0001	_	0	0	0	1	
28 = m8' m4' m2 m1' leap'	0010	0	1	0	0	0	
	0010	1	0	1	0	0	
	0011	_	0	0	0	1	
29 = m8' m4' m2 m1' leap	0100	_	0	0	1	0	
	0101	_	0	0	0	1	
	0110	_	0	0	1	0	
$\square$ 30 = m8' m4 m1' + m8 m1	0111	_	0	0	0	1	
	1000	_	0	0	0	1	
	1001	_	0	0	1	0	
□ 31 = m8′ m1 + m8 m1′	1010	_	0	0	0	1	
	1011	_	0	0	1	0	
	1100	_	0	0	0	1	
Contranalate to C o D or D o C	1101	_	_	_	_	_	
Can translate to 5-0-P of P-0-5	111–	_	_	_	_	_	

month

leap

28

29

30

31

## Leap year flag

- Determine value of leap year flag given the year
  - □ For years after 1582 (Gregorian calendar reformation),
  - leap years are all the years divisible by 4,
  - except that years divisible by 100 are not leap years,
  - but years divisible by 400 are leap years.
- Encoding the year:
  - binary easy for divisible by 4, but difficult for 100 and 400 (not powers of 2)
  - BCD easy for 100, but more difficult for 4, what about 400?
- Parts:
  - construct a circuit that determines if the year is divisible by 4
  - construct a circuit that determines if the year is divisible by 100
  - construct a circuit that determines if the year is divisible by 400
  - combine the results of the previous three steps to yield the leap year flag

#### Activity: divisible-by-4 circuit

Use BCD encoded year instead of binary encoded year.

- YM8 YM4 YM2 YM1 YH8 YH4 YH2 YH1 YT8 YT4 YT2 YT1 YO8 YO4 YO2 YO1
- Idea to get a divisible-by-4-circuit
  - 00-04-08, 12-16, 20
  - □ if tens digit is even, then divisible by 4 if ones digit is 0, 4, or 8
  - if tens digit is odd, then divisible by 4 if the ones digit is 2 or 6.

#### Boolean Expression

- YT1' (Y08' Y04' Y02' Y01' + Y08' Y04 Y02' Y01' + Y08 Y04' Y02' Y01') + YT1 (Y08' Y04' Y02 Y01' + Y08' Y04 Y02 Y01')
- Simplified Boolean Expression
  - Digits with values of 10 to 15 will never occur
  - YT1' YO2' YO1' + YT1 YO2 YO1'

#### Divisible-by-100 and divisible-by-400 circuits

Divisible-by-100 just requires checking that all bits of two low-order digits are all 0:

YT8' YT4' YT2' YT1' • YO8' YO4' YO2' YO1'

 Divisible-by-400 combines the divisible-by-4 (applied to the thousands and hundreds digits) and divisible-by-100 circuits

(YM1' YH2' YH1' + YM1 YH2 YH1')

• (YT8' YT4' YT2' YT1' • YO8' YO4' YO2' YO1' )

#### Combining to determine leap year flag

Label results of previous three circuits: D4, D100, and D400

 $leap_year_flag = D4 (D100 \bullet D400')'$ 

= D4 • D100' + D4 • D400

= D4 • D100' + D400

#### Implementation of leap year flag

