Combinational logic design case studies

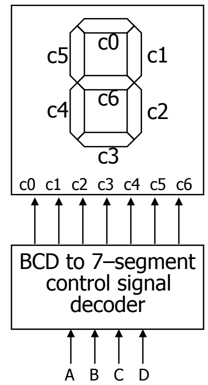
- Arithmetic circuits
  - integer representations
  - addition/subtraction
  - arithmetic/logic units
- General design procedure
- Case studies
  - BCD to 7-segment display controller
  - Leap-year flag calculator

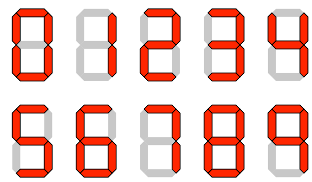
General design procedure for combinational logic

- 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
  - programmable logic: FPGA, ROM, PAL, PLA
  - mux, decoder and OR-gate
  - discrete gates
- 4. Follow implementation procedure
  - HDL to be synthesized, K-maps for two-level logic, multi-level logic tools
  - design tools and hardware description language (e.g., Verilog) are crucial

BCD to 7-segment display controller

- Understanding the problem
  - input is a 4 bit bcd digit (A, B, C, D)
  - output is the control signals for the display (7 outputs C0 – C6)
- Block diagram





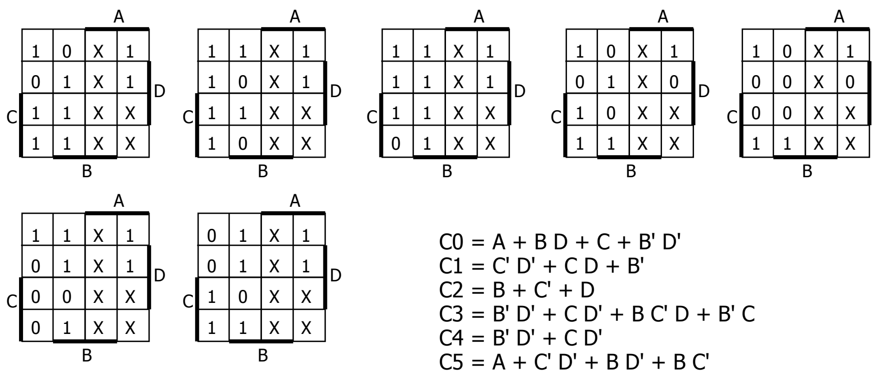
Formalize the problem

- Truth table
  - show don't cares
- Choose implementation target
  - if ROM, we are done
  - don't cares imply PAL/PLA may be attractive
- Follow implementation procedure
  - minimization using K-maps

A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	–	–	–	–	–	–	–	–
1	1	–	–	–	–	–	–	–	–	–

Implementation as minimized sum-of-products

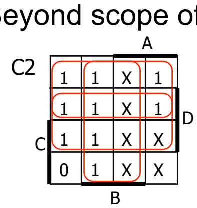
- 15 unique product terms when minimized individually

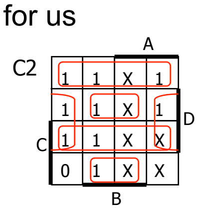


C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

Multiple-Output Logic Minimization

- Can do better
  - 9 unique product terms (instead of 15)
  - share terms among outputs
  - each output not necessarily in minimized form
- Beyond scope of class – tools do this for us

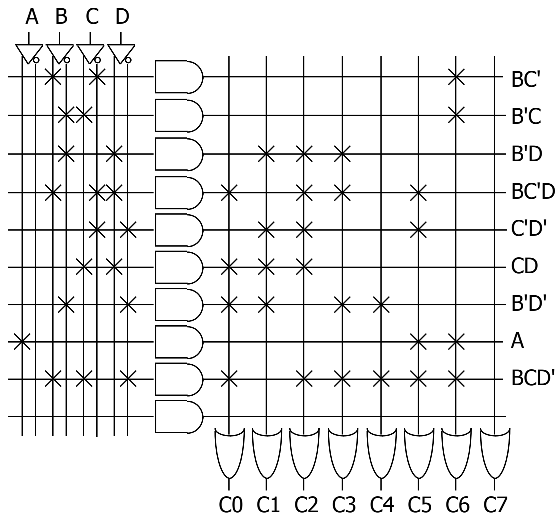




C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

C0 = B C' D + C D + B' D' + B C D' + A
C1 = B' D + C' D' + C D + B' D'
C2 = B' D + B C' D + C' D' + C D + B C D'
C3 = B C' D + B' D + B' D' + B C D'
C4 = B' D' + B C D'
C5 = B C' D + C' D' + A + B C D'
C6 = B' C + B C' + B C D' + A

PLA implementation



PAL implementation vs. Discrete gate implementation

- Limit of 4 product terms per output
  - decomposition of functions with larger number of terms
  - do not generally share terms in PAL anyway (although there are exceptions)

C2 = B + C' + D

C2 = B' D + B C' D + C' D' + C D + B C D'

C2 = B' D + B C' D + C' D' + W ← need another input and another output
W = C D + B C D'

- decompose into multi-level logic (hopefully with CAD support)
  - find common sub-expressions among functions

C0 = C3 + A' B X' + A D Y
C1 = Y + A' C5' + C' D' C6
C2 = C5 + A' B' D + A' C D
C3 = C4 + B D C5 + A' B' X'
C4 = D' Y + A' C D'
C5 = C' C4 + A Y + A' B X
C6 = A C4 + C C5 + C4' C5 + A' B' C

X = C' + D'
Y = B' C'

Activity: divisible-by-4 circuit

- BCD coded year (digits for thousands, hundreds, tens, and ones)
  - YM8 YM4 YM2 YM1 , YH8 YH4 YH2 YH1 , YT8 YT4 YT2 YT1 , YO8 YO4 YO2 YO1

Activity: divisible-by-4 circuit

- BCD coded year (digits for thousands, hundreds, tens, and ones)
  - YM8 YM4 YM2 YM1 , YH8 YH4 YH2 YH1 , YT8 YT4 YT2 YT1 , YO8 YO4 YO2 YO1
- Only need to look at low-order two digits of the year
  all years ending in 00, 04, 08, 12, 16, 20, etc. are divisible by 4
  - 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
- Translates into the following Boolean expression:
  - YT1’ (YO8’ YO4’ YO2’ YO1’ + YO8’ YO4 YO2’ YO1’ + YO8 YO4’ YO2’ YO1’)
    + YT1 (YO8’ YO4’ YO2 YO1’ + YO8’ YO4 YO2 YO1’)
- Digits with values from 10 to 15 will never occur, simplify further to yield:
  - YT1’ (YO == 0 | YO == 4 | YO == 8 | YO == 12)
    + YT1 (YO == 2 | YO == 6 | YO == 10 | YO == 14)
  - YT1’ YO2’ YO1’ + YT1 YO2 YO1’

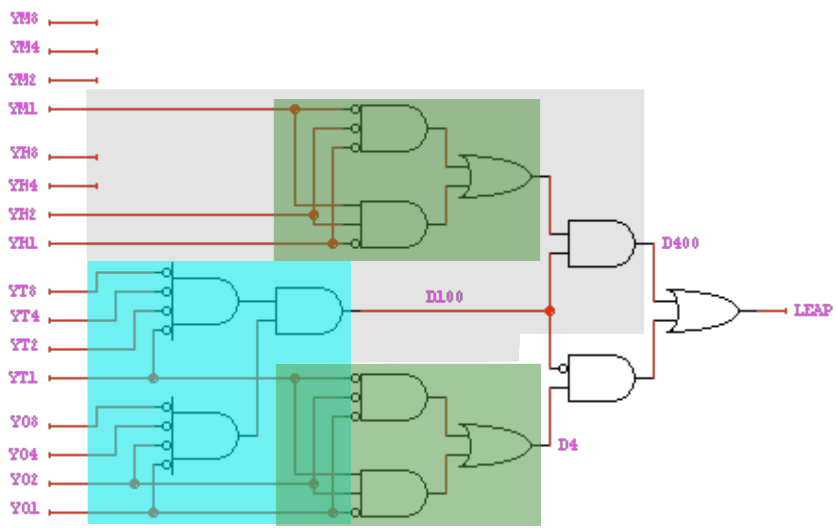
Divisible-by-100 and divisible-by-400 circuits

- 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’)

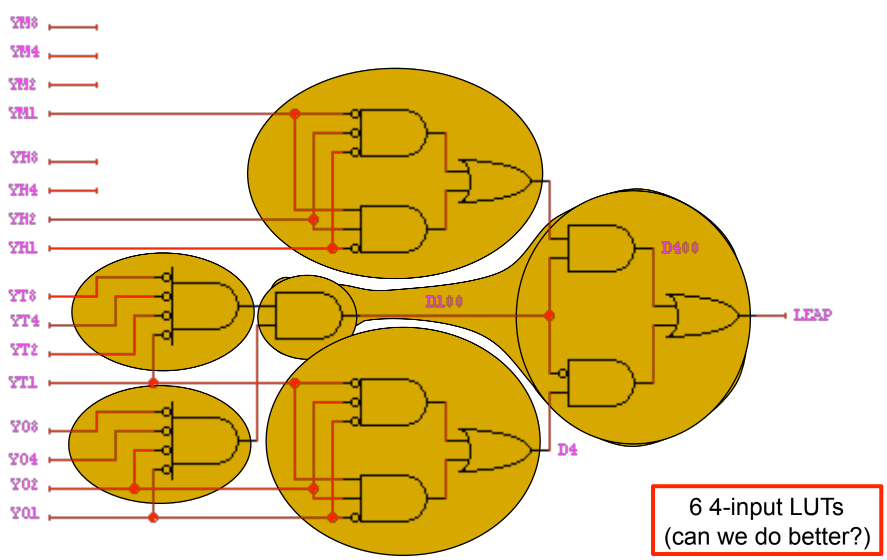
Combining to determine leap year flag

- Label outputs of the previous three circuits: D4, D100, and D400
  - leap_year_flag = D4 • D100’ + D400

Implementation of leap year flag



Implementation of leap year flag in FPGA



Summary for examples of combinational logic

- Combinational logic design process
  - formalize problem: encodings, truth-table, equations
  - choose implementation technology (FPGA, ROM, PAL, PLA, discrete gates)
  - implement by following the design procedure for that technology
- Binary number representation
  - positive numbers the same
  - difference is in how negative numbers are represented
  - 2s complement easiest to handle: one representation for zero, slightly complicated complementation, simple addition
- Circuits for binary addition
  - basic half-adder and full-adder
  - carry lookahead logic
  - carry-select