Midterm Topics (11/29)

- Chapters Appendix A and Section 5,2,3,4
- Introduction
  - Gates (CMOS switch model)
  - Number Systems
- Boolean Logic
- Logic Minimization (Algebra, K-maps)
- Bubble Propagation
- Timing behavior of combination logic (gate delay)
- 2-level Static Hazards
- Combination Logic System Design (Multi-Level)
- Boolean Logic Implementation
  - MUX
  - PAL, PLA
  - ROM
  - Wired Logic, Pass Gates, and Tri-State

Combinational Logic System Design

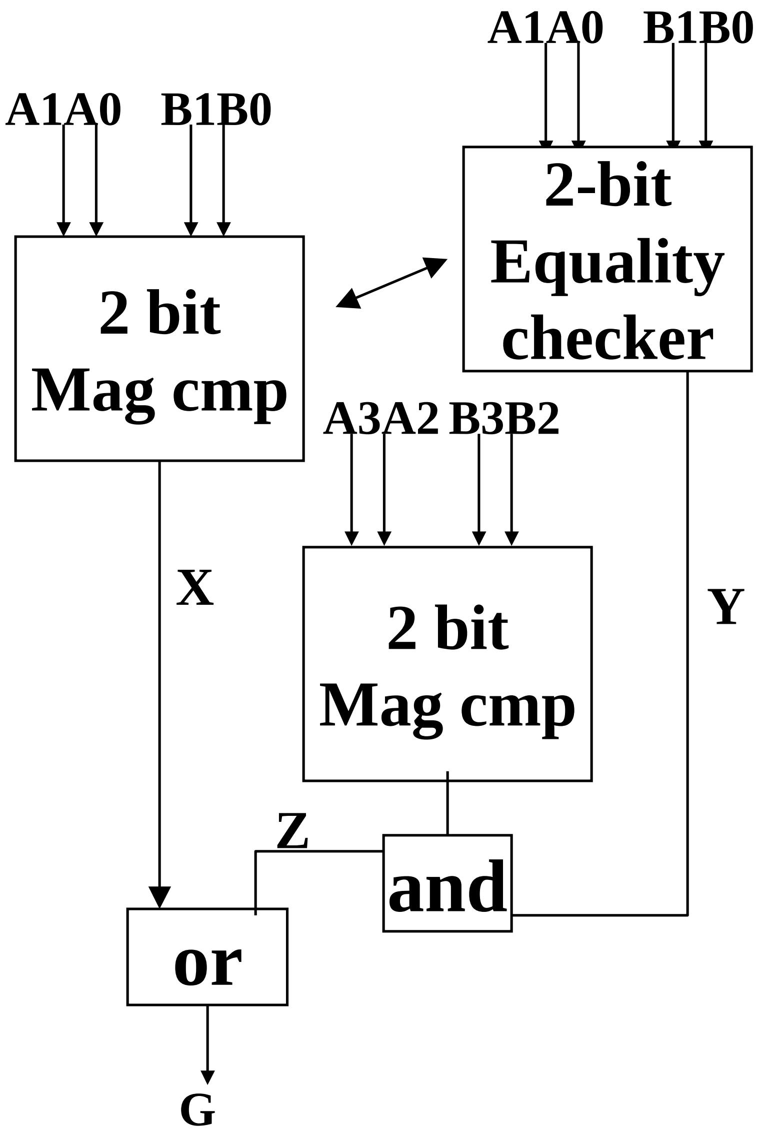
- 1. Understand the problem
  - what is the circuit supposed to do?
  - write down inputs (data, control) and outputs
  - draw block diagram or other picture
- 2. Formulate the problem using a suitable design representation
  - truth table or waveform diagram are typical
  - may require encoding of symbolic inputs and outputs
  - Pseudo-code definition of the block or system
- 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)

System Design Example

- **Floating Point Magnitude Comparator**
  - **G = 1 if A>B for 4-bit floating point**
- **Function of 8 inputs!!**
  - **A = <A3,A2,A1,A0>**
  - **B = <B3,B2,B1,B0>**

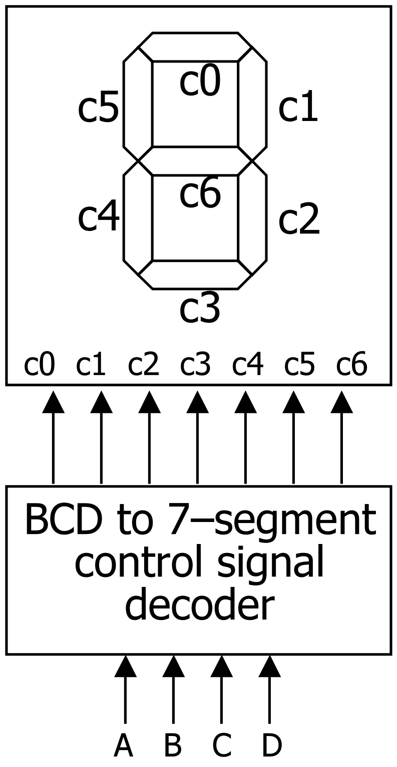
<u>Pseudo-code</u>

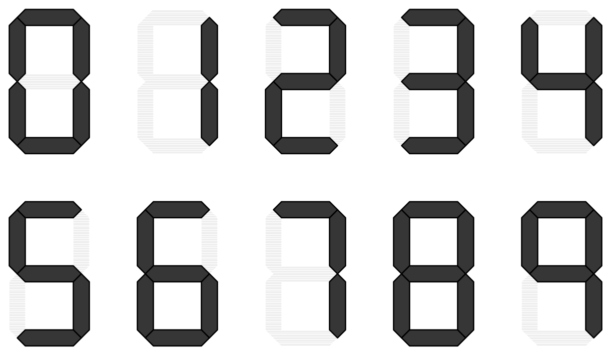
```
if (X = (A1A0 > B1B0)) G = 1
if (Y = A1A0 == B1B0)
        if (Z = A-man > B-man) G = 1
else G = 0
So G = X + YZ

if (X = (A1A0 > B1B0)) G = 1
else if (Y = !(B1B0 > A1A0))
        if (Z = A-man > B-man) G = 1
else G = 0
So G = X + X'YZ
```



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





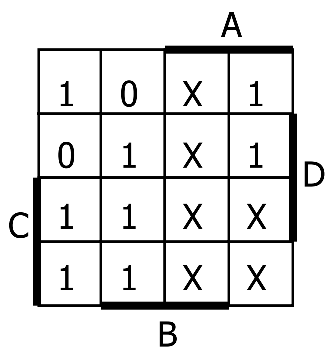
Formalize the problem

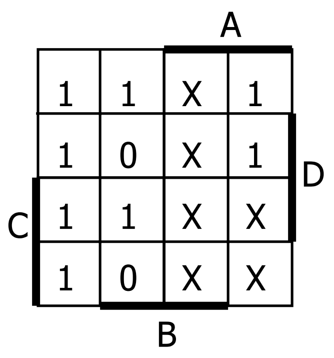
- Truth table
  - show don't cares
- Choose implementation target
  - if ROM, we are done
  - don't cares imply PAL/PLA may be attractive
- Follow implementation procedure
  - minimization using K-maps

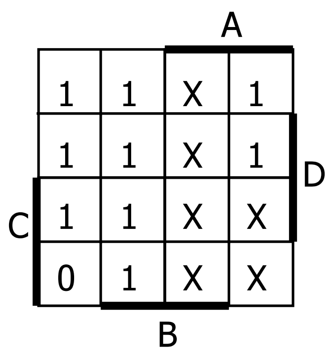
A	B	C	D	C0	C1	C2	C3	C4	C5	C6
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1
1	0	1	–	–	–	–	–	–	–	–
1	1	–	–	–	–	–	–	–	–	–

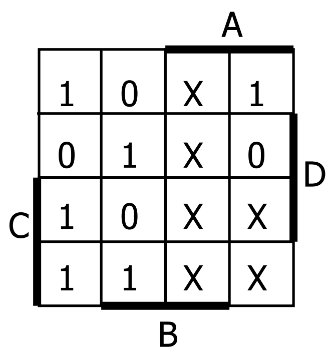
Implementation as minimized sum-of-products

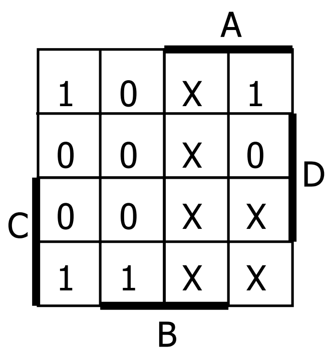
❑ 15 unique product terms when minimized individually

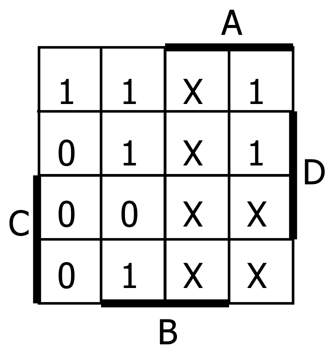


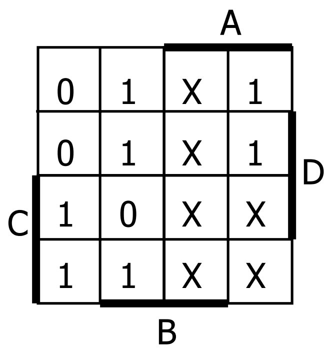










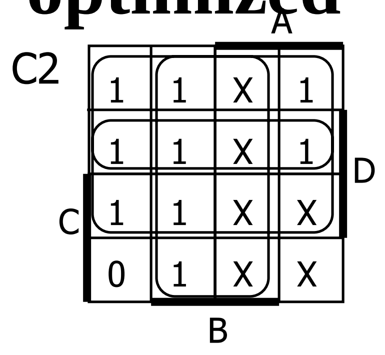


C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = A + B + C' + D
C3 = B' D' + C D' + B C' D + B' C
C4 = B' D' + C D'
C5 = A + C' D' + B D' + B C'
C6 = A + C D' + B C' + B' C

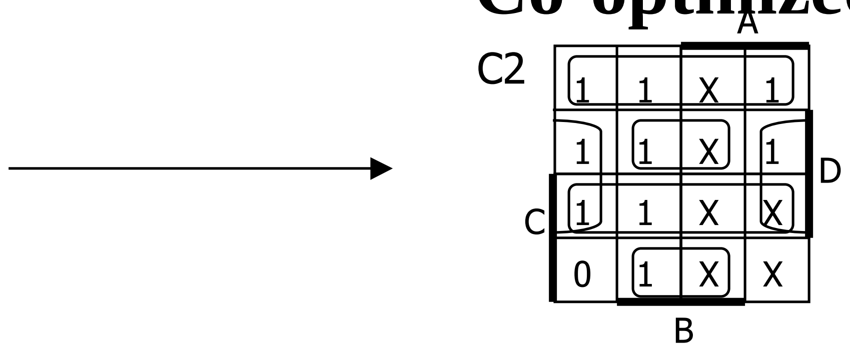
Implementation as minimized S-o-P (cont'd)

- Can do better
  - 9 unique product terms (instead of 15)
  - share terms among outputs
  - each output not necessarily in minimized form – co-optimization

optimized



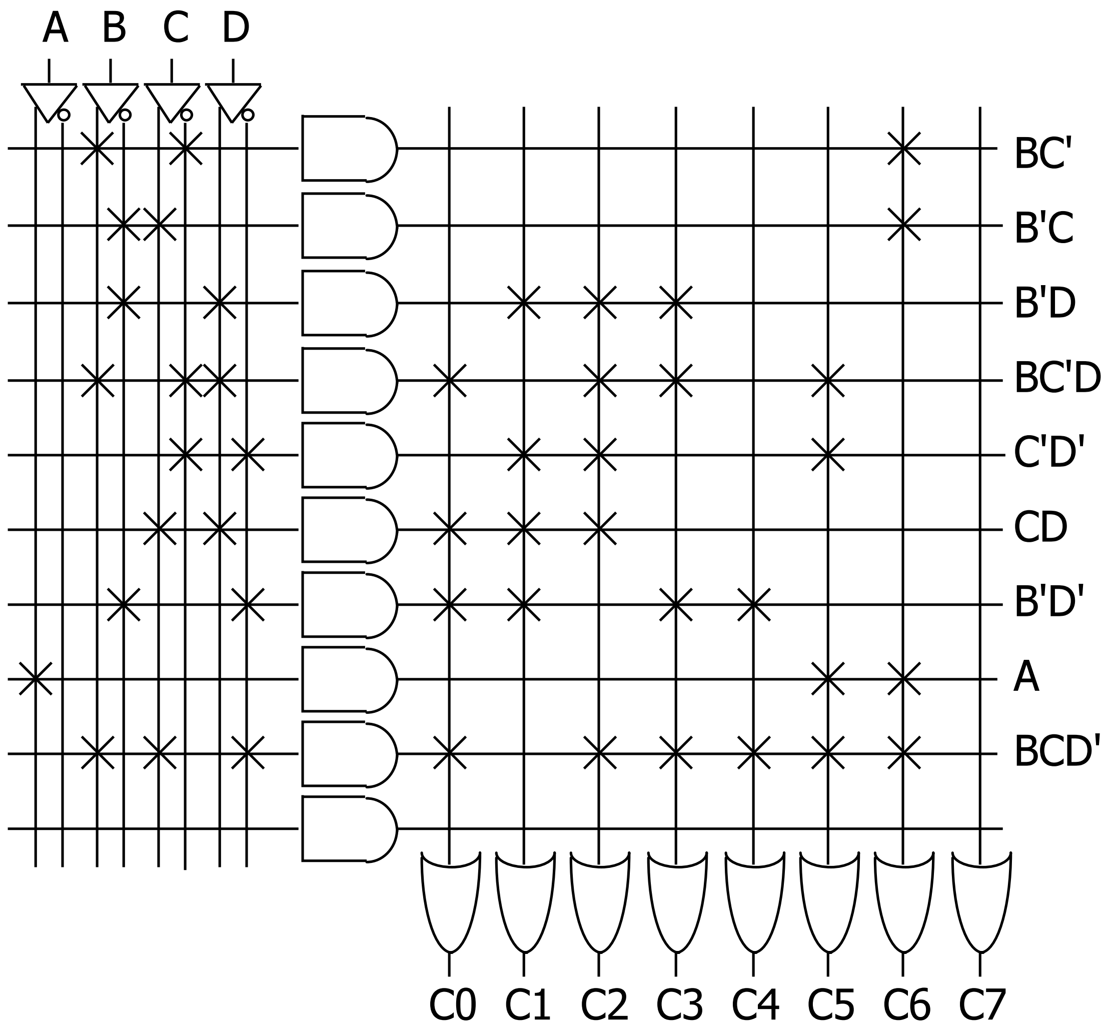
Co-optmized



C0 = A + B D + C + B' D'
C1 = C' D' + C D + B'
C2 = A+ 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

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

    C2 = A + 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
    W = C D + B C D'

    need another input and another output

  - 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'

Alternative Circuit Styles

- Alternative Implementation Methods
  - Pass Gate (Transmission Gate)
  - Tri-state and Busses
  - Wired Logic (Open Collector)
  - MSI, SSI Device Symbols

Pass Gate

CMOS Pass Gate

if S is Hi

Then A and F are shorted

and B and F are disconnected

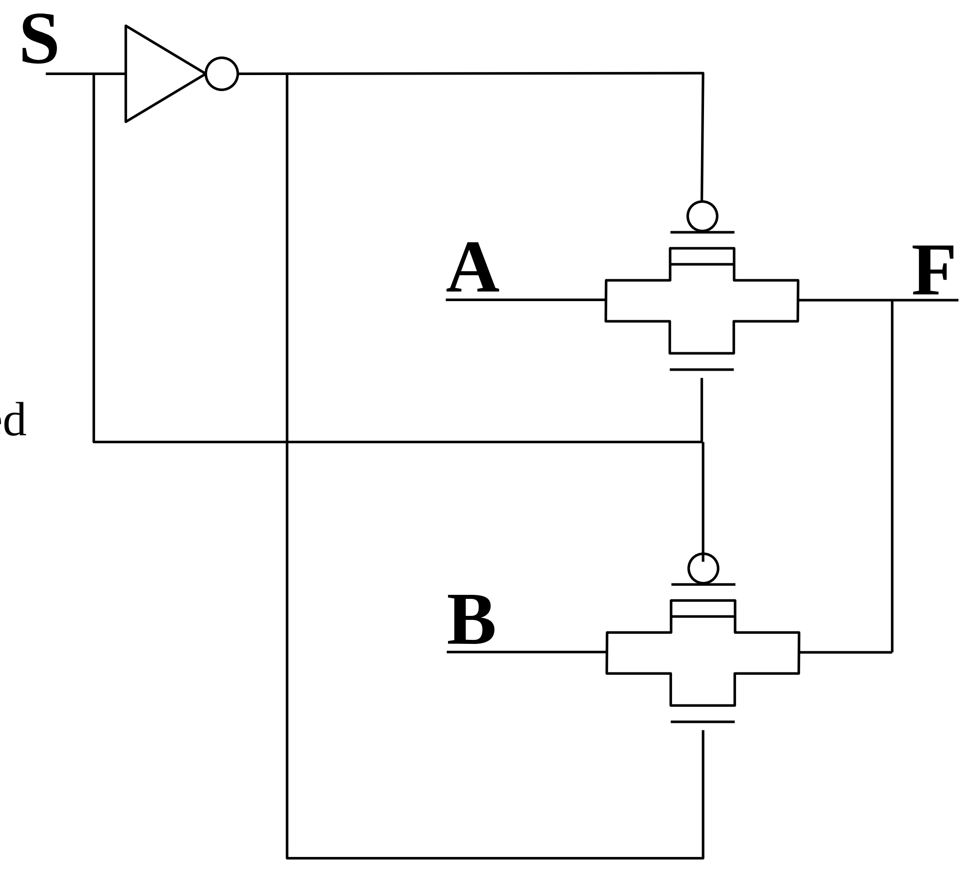
If S is Lo

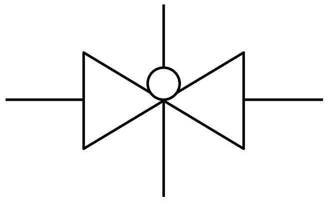
Then A and F are disconnected

and B and F are shorted

What function does this perform?

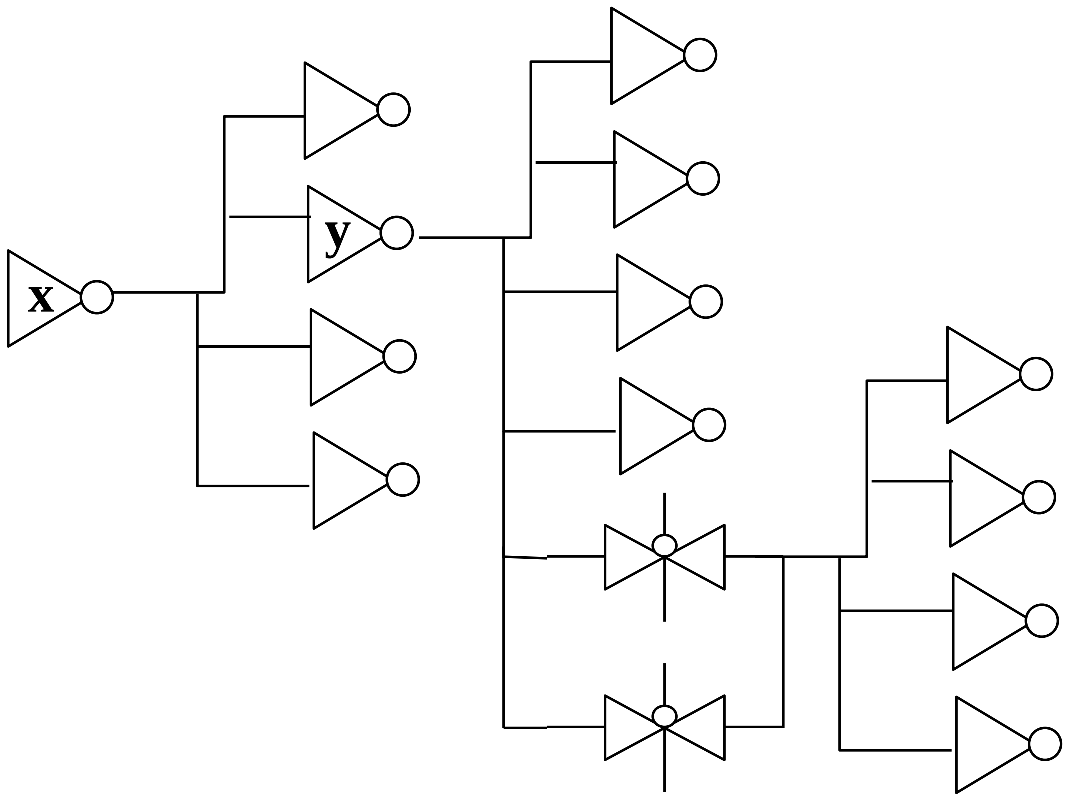
Why not do everything this way?




symbol

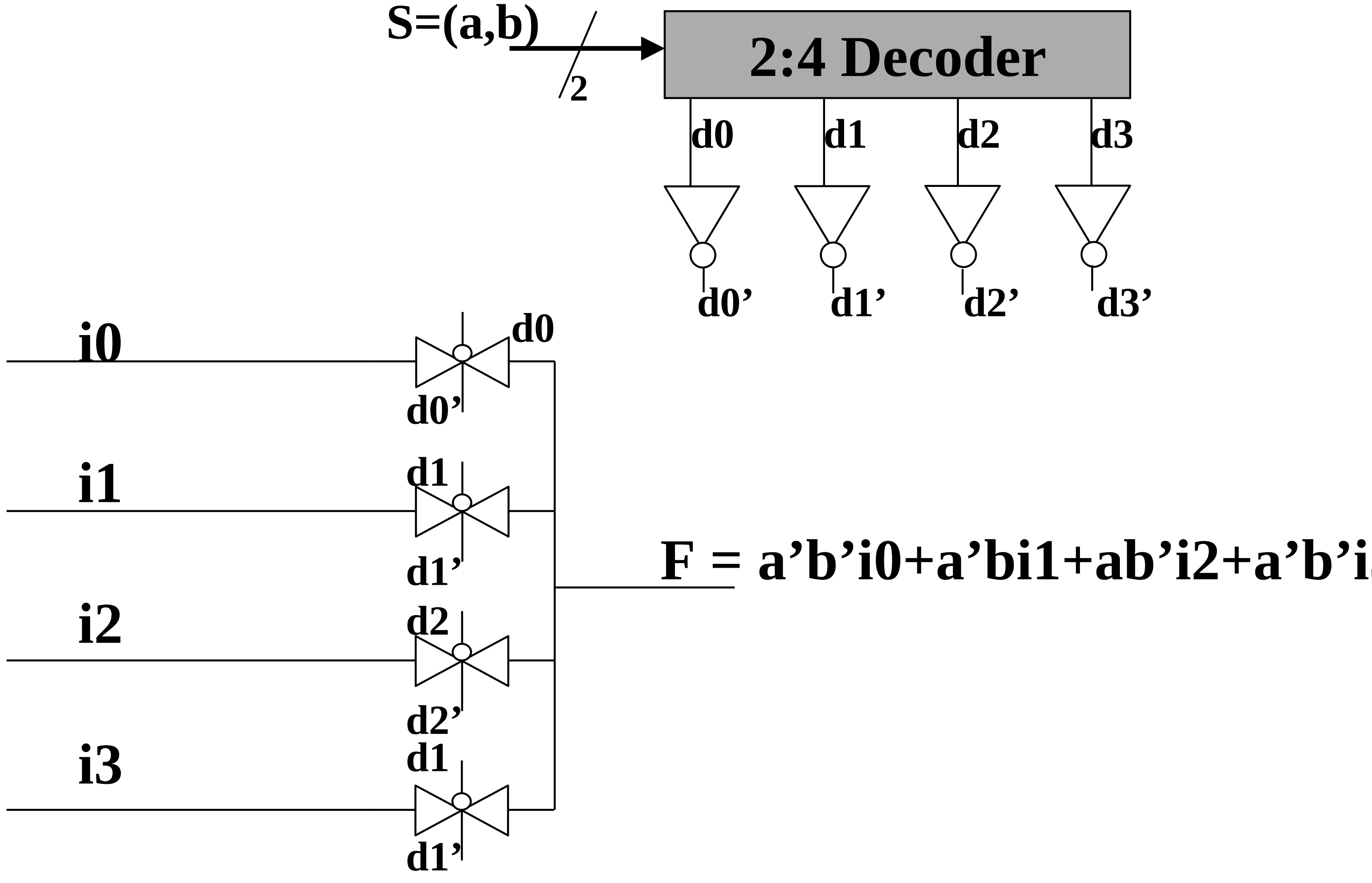
Counting Fanout w/ Pass Gates

Fanout of x is 4



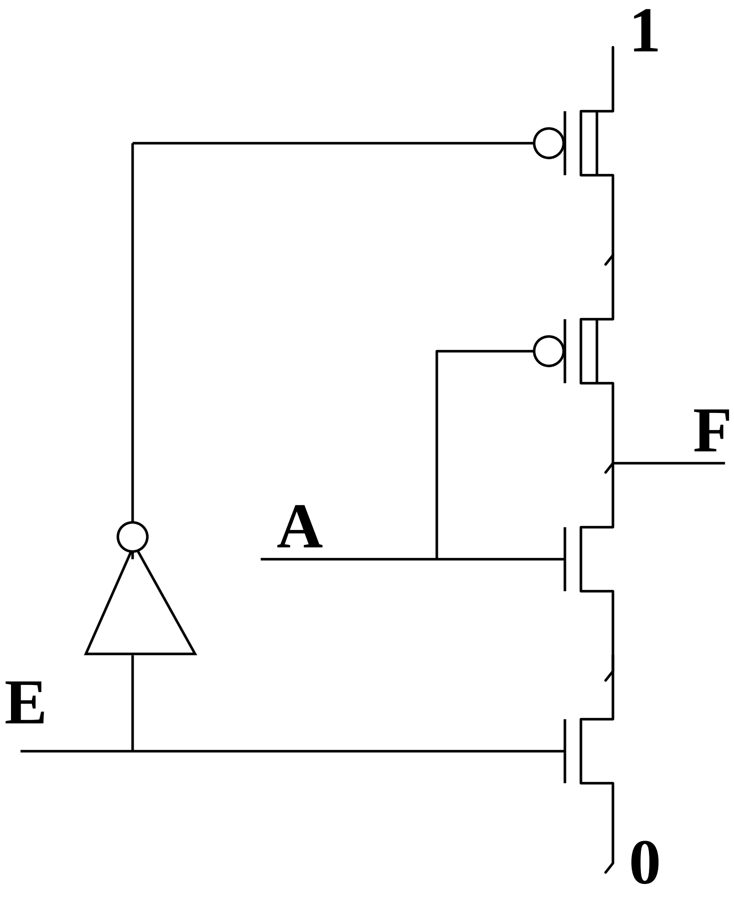
Fanout of y is 8+

Mux from Decoder and Pass Gates



F = a'b'i0+a'bi1+ab'i2+a'b'i3

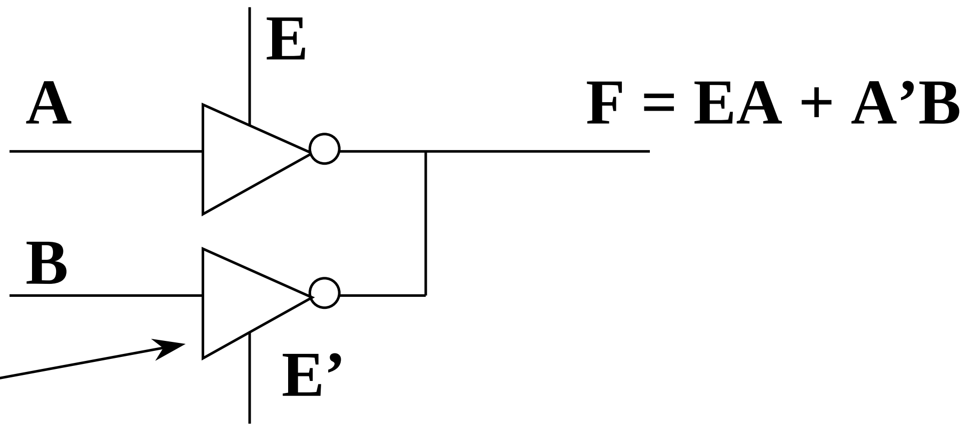
Tri-State Logic



<u>0 = False, 1 = True, z = Hi Impedance</u>

```
If (!E)
        F = A'
else
        F = z
```

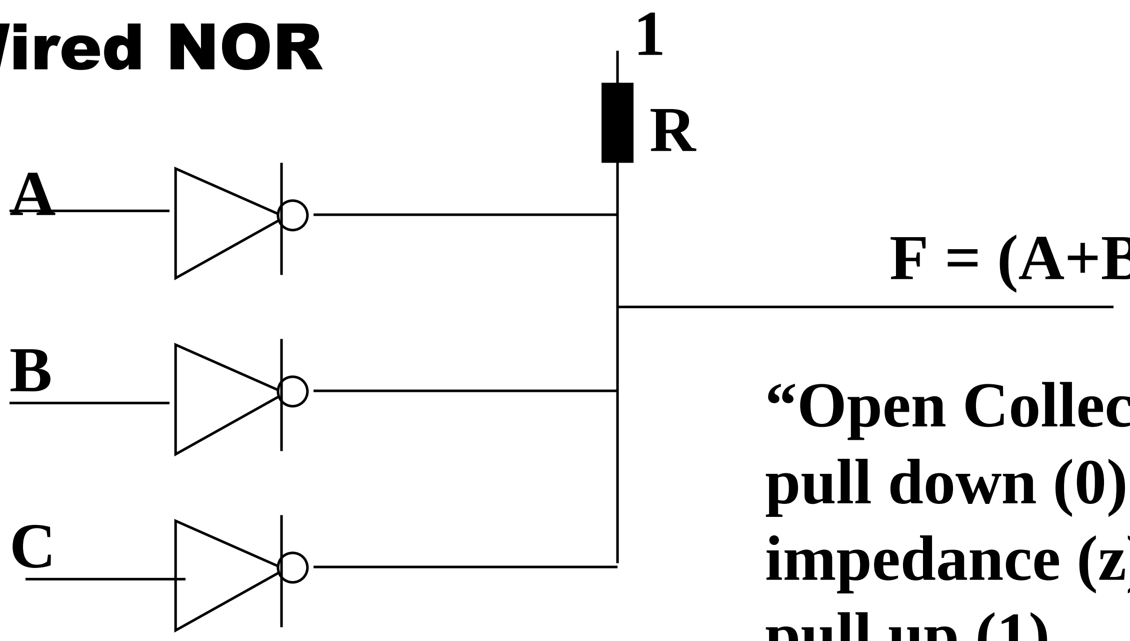
Advantage: support multiple drivers without giving up gain



Tri State Buffer

Wire Logic

Wired NOR



"Open Collector" drivers can pull down (0) or go to high impedance (z), but they can't pull up (1).

- **Used extensively in array devices like RAM/ROM. Must carefully design for speed of charge/discharge. DC power loss when driven low**
- **Also used for Busses on a PC Board. Pull Down's are faster than pull-ups (nMOS is faster than pMOS)**

Number systems

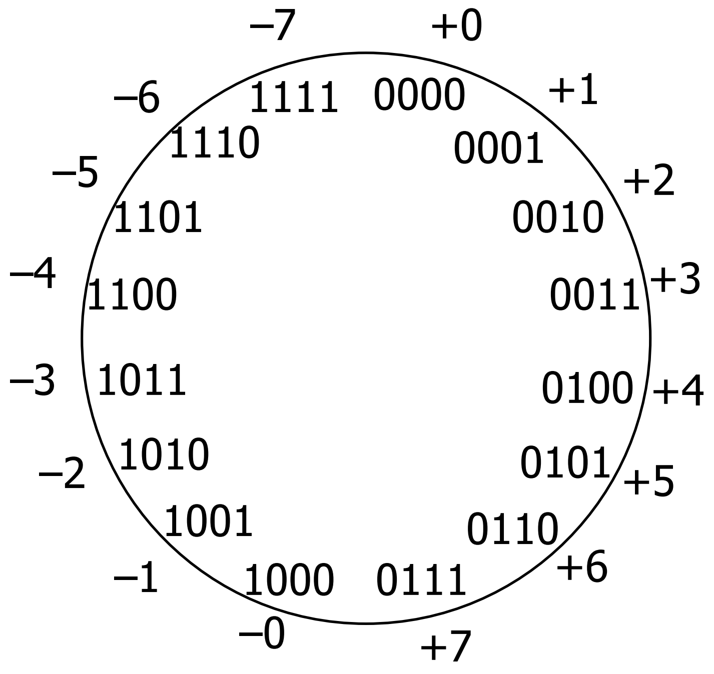
- Representation of positive numbers is the same in most systems
- Major differences are in how negative numbers are represented
- Representation of negative numbers come in three major schemes
  - sign and magnitude
  - 1s complement
  - 2s complement
- Assumptions
  - we'll assume a 4 bit machine word
  - 16 different values can be represented
  - roughly half are positive, half are negative

Sign and magnitude

- One bit dedicate to sign (positive or negative)
  - sign: 0 = positive (or zero), 1 = negative
- Rest represent the absolute value or magnitude
  - three low order bits: 0 (000) thru 7 (111)
- Range for n bits
  - +/– 2n–1 –1 (two representations for 0)
- Cumbersome addition/subtraction
  - must compare magnitudes to determine sign of result
- Discontinuity between + and –
  - Can't use normal arithmetic

0 100 = + 4

1 100 = – 4

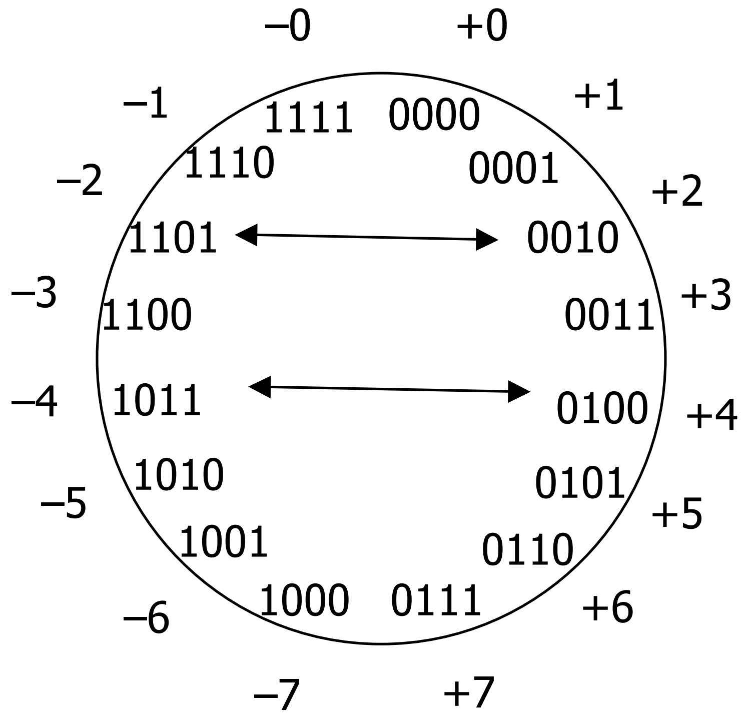


1s complement

- Like S-M but reverse order of negative numbers
- If N is a positive number, then the negative of N (its 1s complement or N') is N' = $(2^n – 1) – N$
  - example: 4-bit 1s complement of 7

$$\begin{array}{lrl} 2^4 - 1 & = & 1111 \\ 7 & = & \underline{0111} \\ & & 1000 \quad = \ -7 \text{ in 1s comp.} \end{array}$$

  - compute by taking bit-wise complement (0111 -> 1000)
  - Add/Sub must account for double zero when



$$\begin{array}{r} 1110 \\ +\ \underline{0010} \\ 1\ 1100 \\ \hookrightarrow 1 \\ \overline{1101} \end{array} \qquad \begin{array}{r} 0010 \\ -\ \underline{0011} \\ 1\ 1111 \\ \hookrightarrow 1 \\ \overline{1110} \end{array}$$

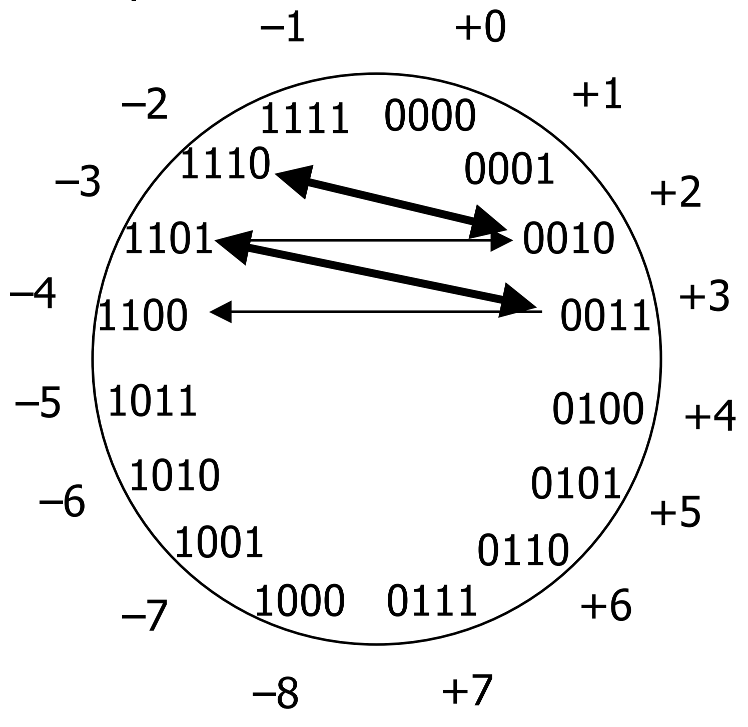
2s complement

- 1s complement with negative numbers shifted one position clockwise
  - only one representation for 0
  - one more negative number than positive number
  - high-order bit can act as sign bit
  - No need to compensate for +/-
  - Compute complement by adding 1 to1’s complement

-1 + 2 = 0 + carry
2 – 3 = 0 - borrow

```
     1111        0010
   + 0010      - 0011
   ------      ------
 1   0001    1   1111
```

Ignore borrow/carry because we changed sign



2s complement (cont'd)

- If N is a positive number, then the negative of N (its 2s complement or N*) is $N^* = (2^n - 1) - N + 1 = 2^n - N$
  - example: 2s complement of 7

$$\begin{array}{lrcr} & 2^4 & = & 10000 \\ \text{subtract} & 7 & = & \underline{\quad 0111} \\ & & & 1001 \ = \text{repr. of } -7 \end{array}$$

  - example: 2s complement of –7

$$\begin{array}{lrcr} & 2^4 & = & 10000 \\ \text{subtract} & -7 & = & \underline{\quad 1001} \\ & & & 0111 \ = \text{repr. of } 7 \end{array}$$

  - shortcut: 2s complement = bit-wise complement + 1
    - 0111 -> 1000 + 1 -> 1001 (representation of -7)
    - 1001 -> 0110 + 1 -> 0111 (representation of 7)

Overflow Cases in 2's Complement

- Adding two numbers of opposite sign – no overflow

```
     y
    0xxx
    1xxx
   yy'   either value of y is okay (y == carry)
```

- Adding two numbers of same sign. Looks like change of sign

```
     y
    1xxx
    1xxx
   1y     y must be 1 (y == carry)

     y
    0xxx
    0xxx
   0y     y must be 0 (y == carry)
```

In all cases

$V = Cn \oplus Cn+1$

Overflow Examples

- Overflow when carry into sign bit position is not equal to carry-out

$$\begin{array}{r} 5 \\ 3 \\ \hline -8 \end{array} \qquad \begin{array}{r} \scriptstyle 0\ 1\ 1\ 1 \\ 0\,1\,0\,1 \\ 0\,0\,1\,1 \\ \hline 1\,0\,0\,0 \end{array}$$

overflow

$$\begin{array}{r} -7 \\ -2 \\ \hline 7 \end{array} \qquad \begin{array}{r} \scriptstyle 1\ 0\ 0\ 0 \\ 1\,0\,0\,1 \\ 1\,1\,1\,0 \\ \hline 1\,0\,1\,1\,1 \end{array}$$

overflow

$$\begin{array}{r} 5 \\ 2 \\ \hline 7 \end{array} \qquad \begin{array}{r} \scriptstyle 0\ 0\ 0\ 0 \\ 0\,1\,0\,1 \\ 0\,0\,1\,0 \\ \hline 0\,1\,1\,1 \end{array}$$

no overflow

$$\begin{array}{r} -3 \\ -5 \\ \hline -8 \end{array} \qquad \begin{array}{r} \scriptstyle 1\ 1\ 1\ 1 \\ 1\,1\,0\,1 \\ 1\,0\,1\,1 \\ \hline 1\,1\,0\,0\,0 \end{array}$$

no overflow