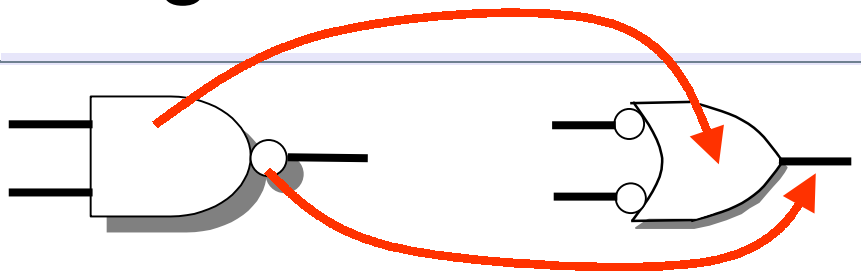
EECS 318 CAD
Computer Aided Design

LECTURE 5:
AOIs,
WITH-SELECT-WHEN,
WHEN-ELSE

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

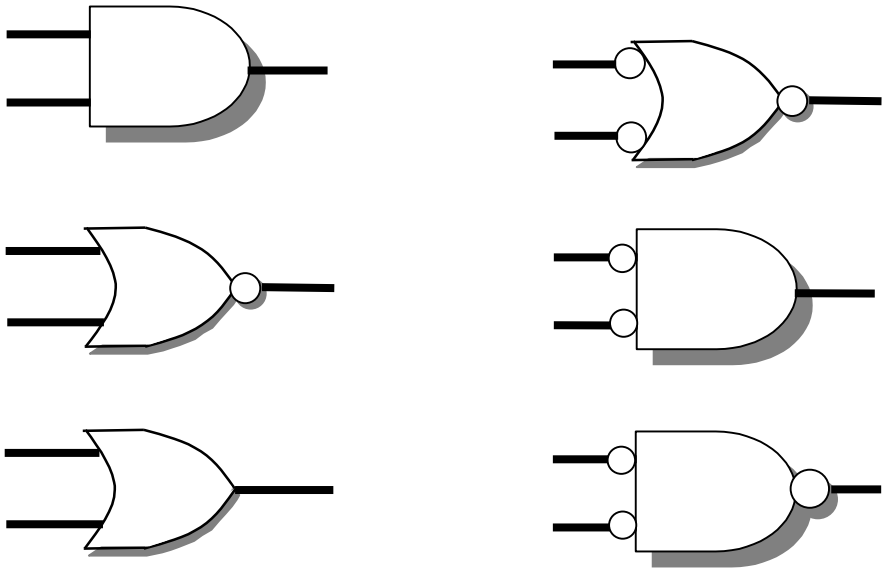
DeMorgan's laws: review



$$\overline{X\ Y} = \overline{X} + \overline{Y}$$

General Rule:

1. **Exchange the AND with OR**
2. **Invert the NOTs**

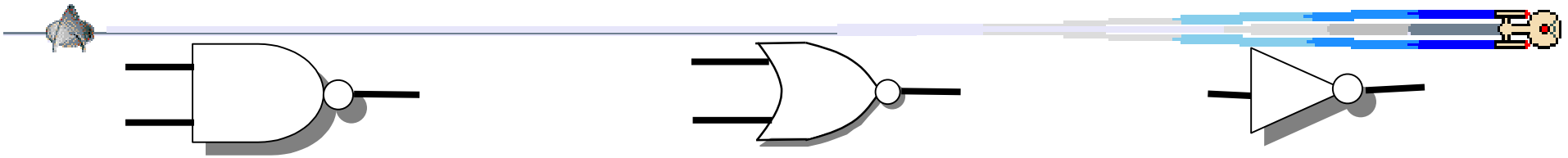


$$X\ Y = \overline{\overline{X} + \overline{Y}}$$

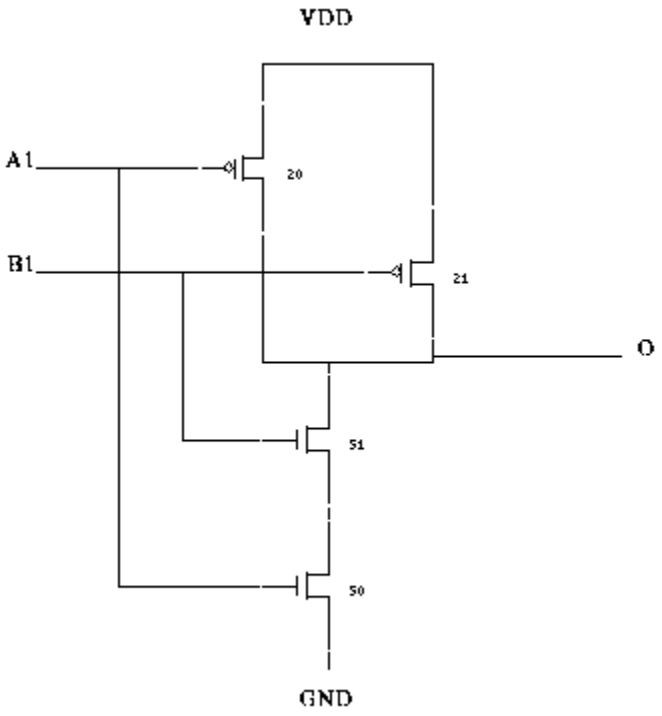
$$\overline{X + Y} = \overline{X}\ \overline{Y}$$

$$X + Y = \overline{\overline{X}\ \overline{Y}}$$

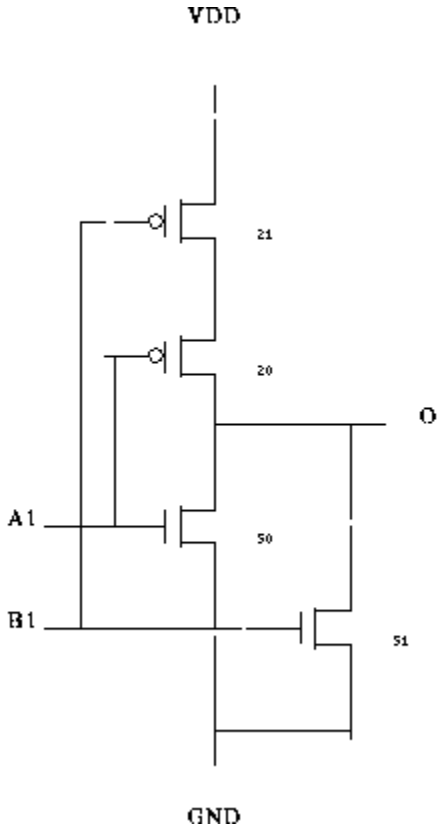
CMOS logic gate: review



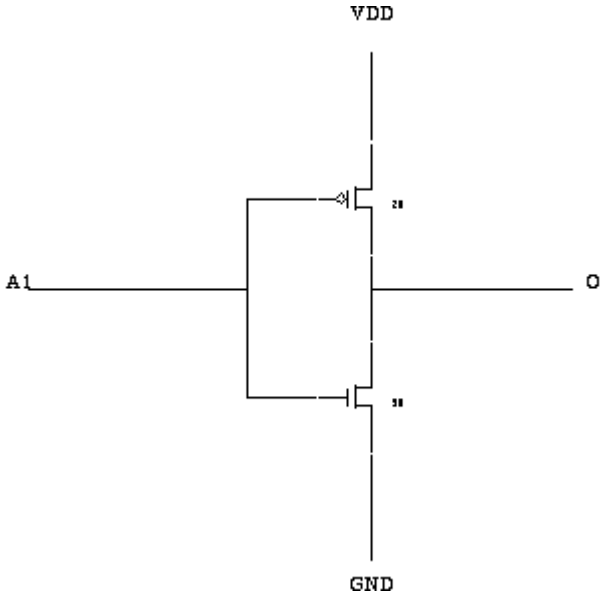
4 transistors



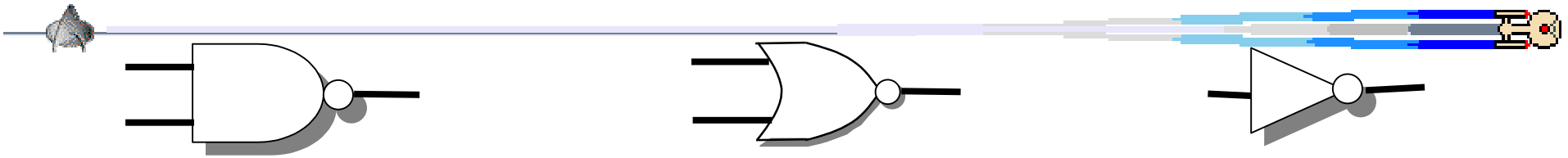
4 transistors

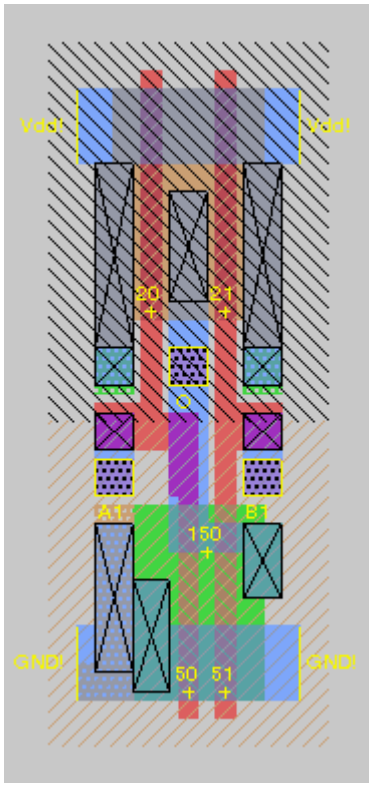


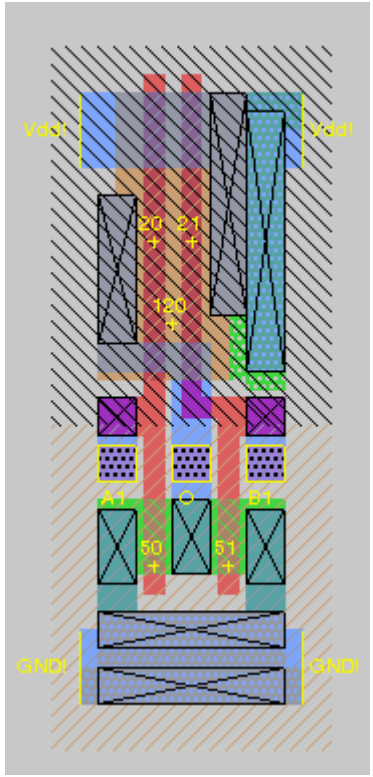
2 transistors

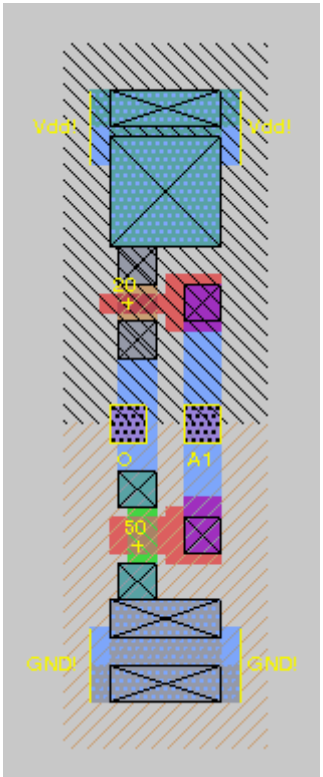


CMOS logic gate: layout sizes (1X output drive)



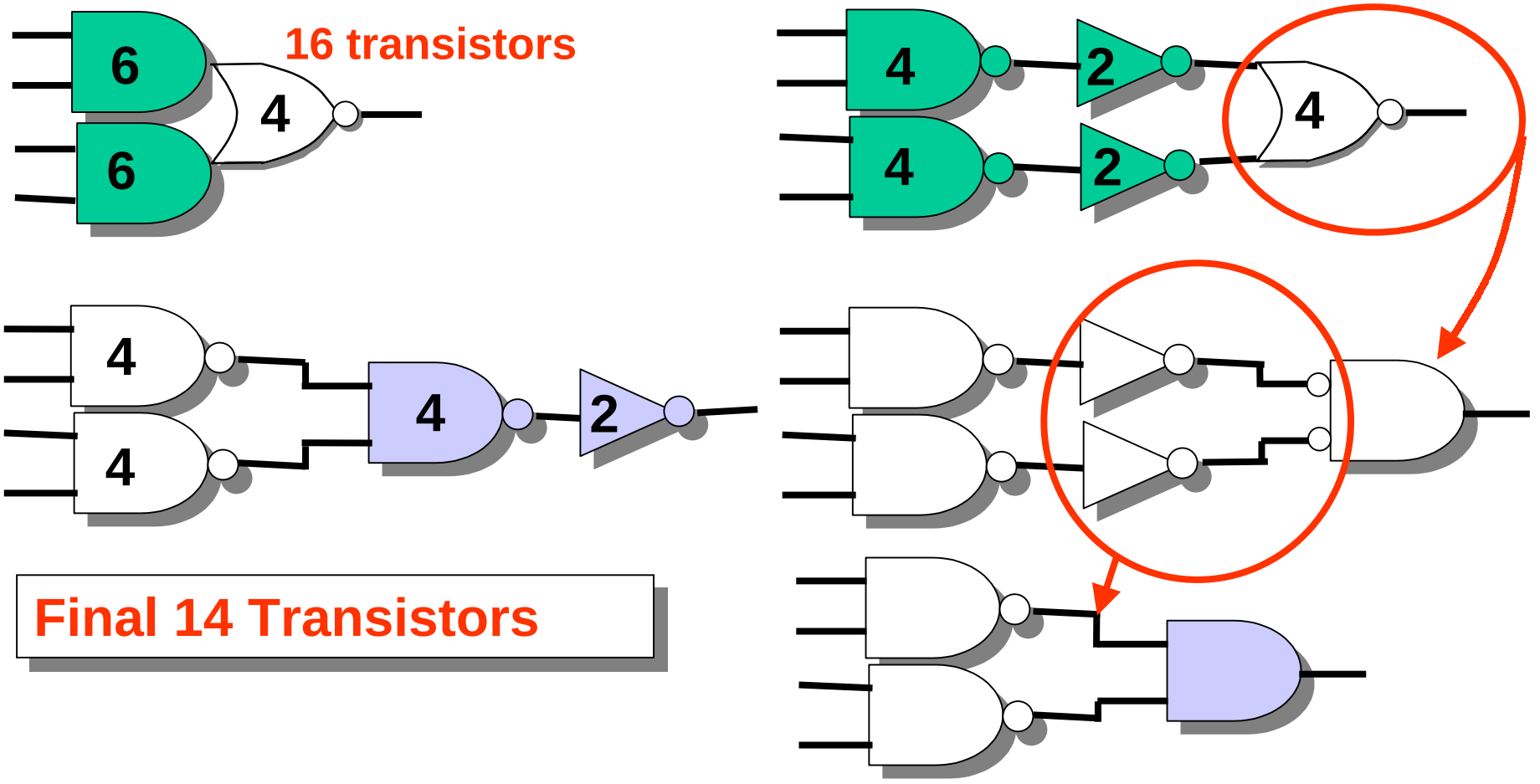






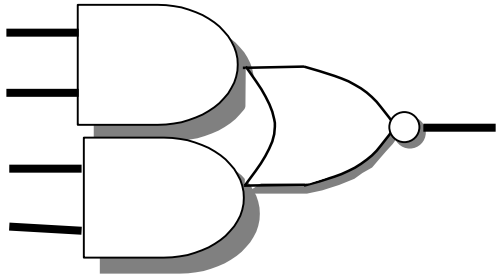
AOI: AND-OR-Invert gates

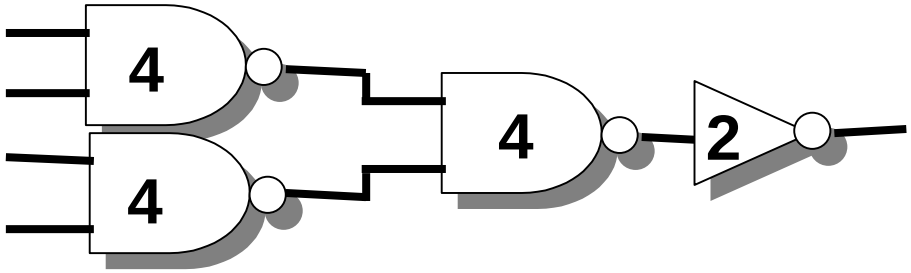
- Suppose you want to transform a circuit to all nands & nots



AOI: AND-OR-Invert gates

- Although, there were no tricks to make AND gates better
- AOIs provide a way at the gate level to use less transistors than separate ANDs and a NORs
- ASIC design logic builds upon a *standard logic cell library*, therefore, do not optimize transistors *only logic gates*
- For example, 2-wide 2-input AOI will only use 8 transistors



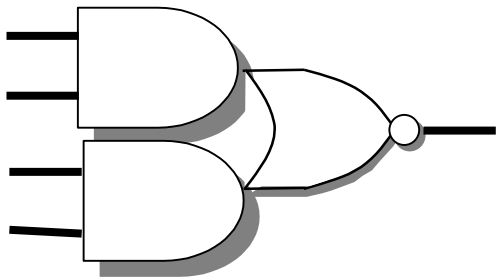


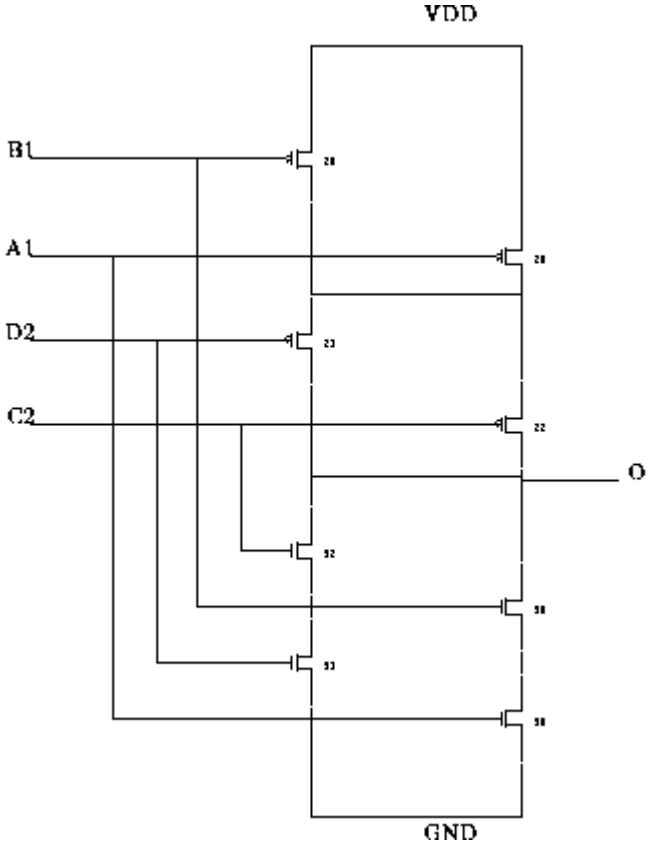
- Whereas 2 ANDs (12 transistors) and 1 NOR (4 transistors) will use a total of 16 transistors *{14 by DeMorgans law}*

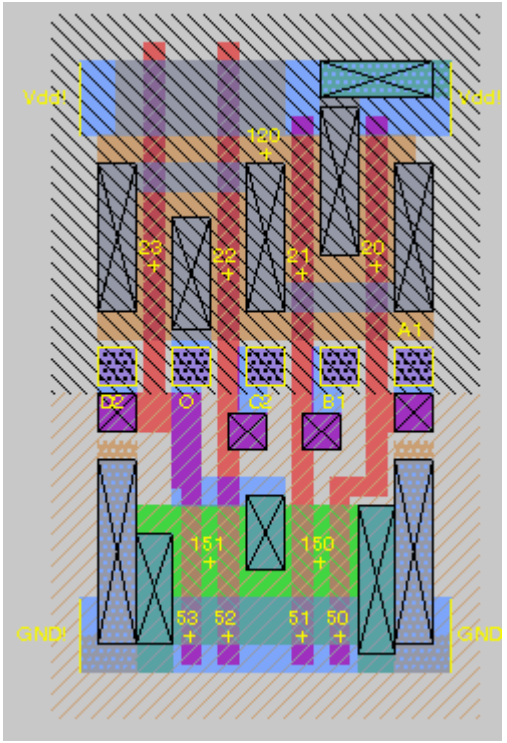
AOI: AND-OR-Invert cmos 2x2 example

- For example, 2-wide 2-input AOI (2x2 AOI)

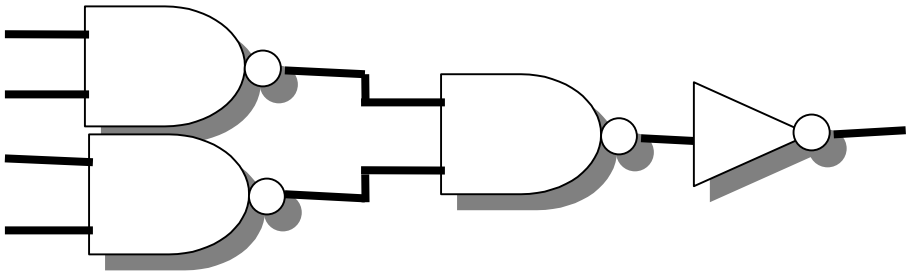
  O <= NOT((D1 AND C1) NOR (B1 AND A1));

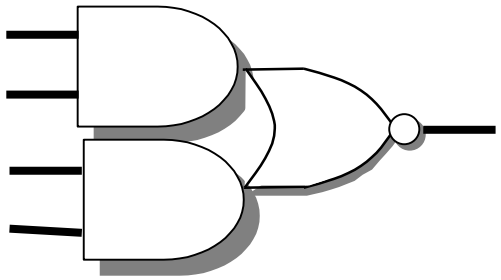


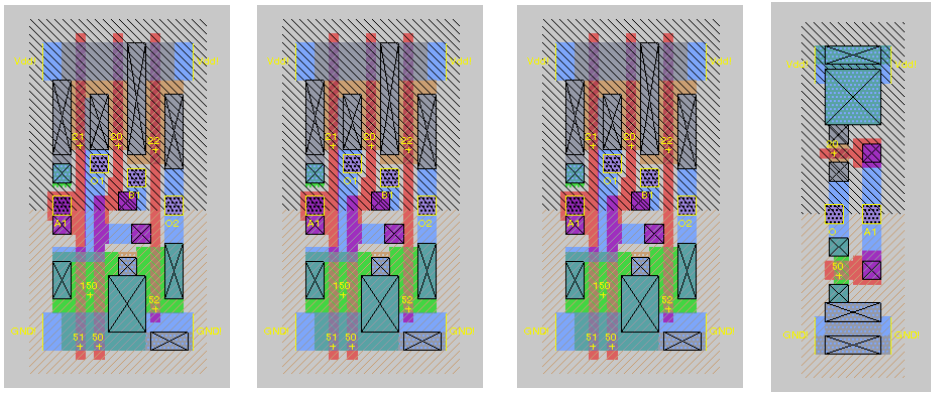


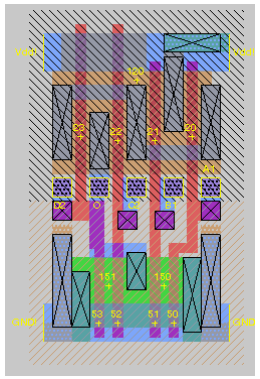


AOI: AND-OR-Invert cmos 2x2 example



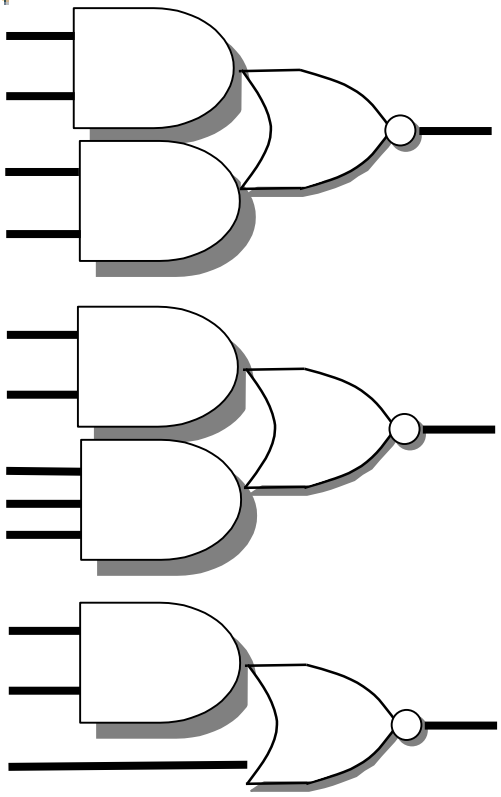






- This means AOIs use less chip area, less power, and delay

AOI: other Standard Cell examples



AOI22 Cell: 2x2 AOI (8 transistors)

Y <= (A AND B) NOR (C AND D);

AOI23 Cell: 2x3 AOI (10 transistors)

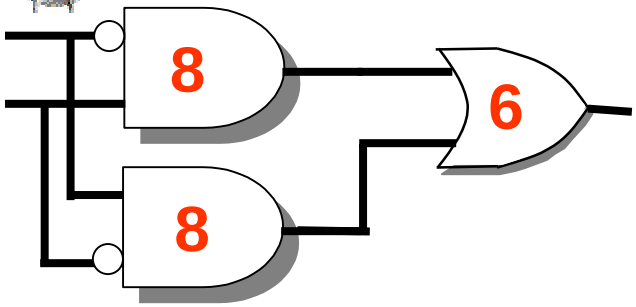
Y <= (A AND B) NOR (C AND D AND E);

AOI21 Cell: 2x1 AOI (6 transistors)

Y <= (A AND B) NOR C;

Total transistors = 2 times # inputs

AOI: XOR implementation

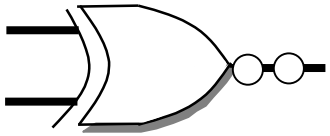


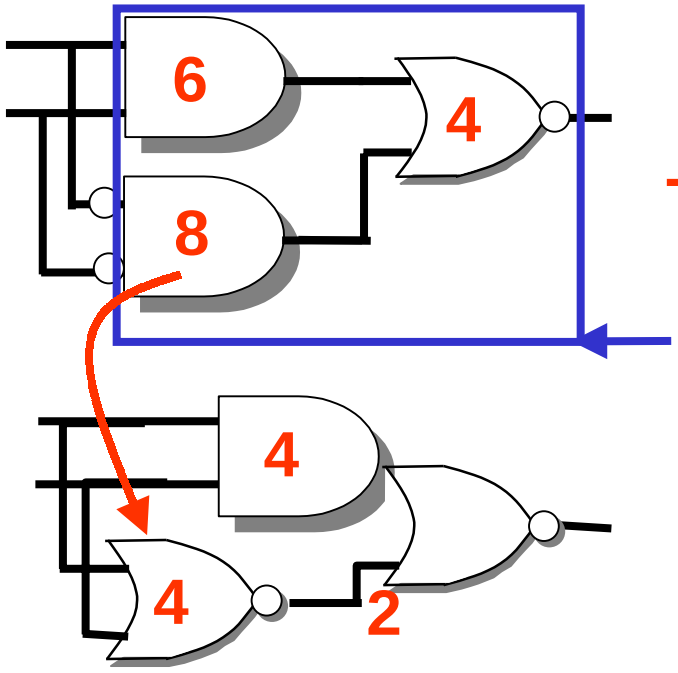
The XOR is not as easy as it appears

Y <= (A AND NOT B) OR (NOT B AND A);

This design uses 22 transistors

Y <= NOT(A XNOR B);





Y <= NOT((A AND B) OR (NOT B AND NOT A));

This newer design uses 18 transistors

But wait, we can exploit the AOI22 structure
now we have 4+4+2+2=12 transistors

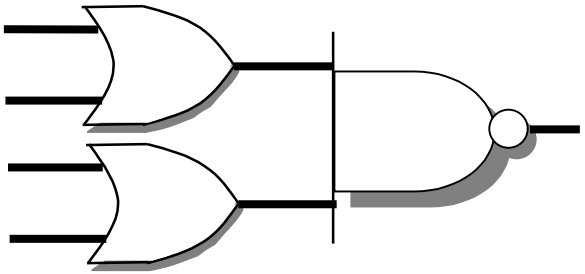
Y <= NOT((A AND B) OR (B NOR A));

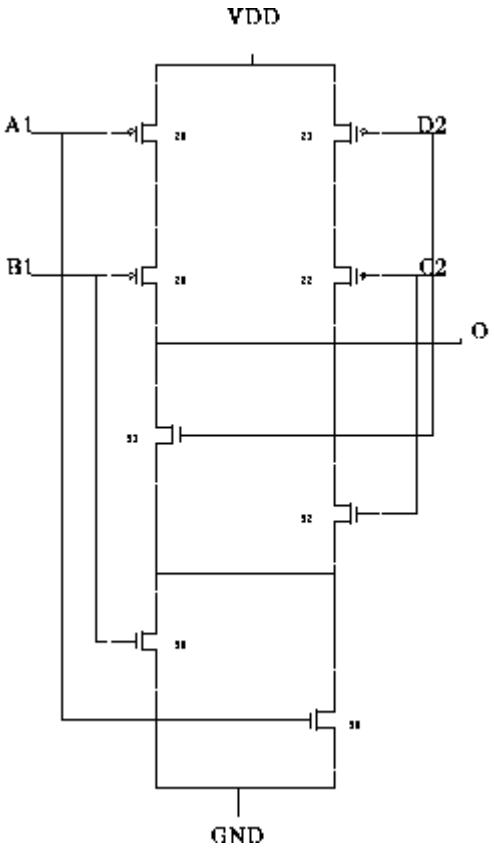
Finally, by applying DeMorgan's law

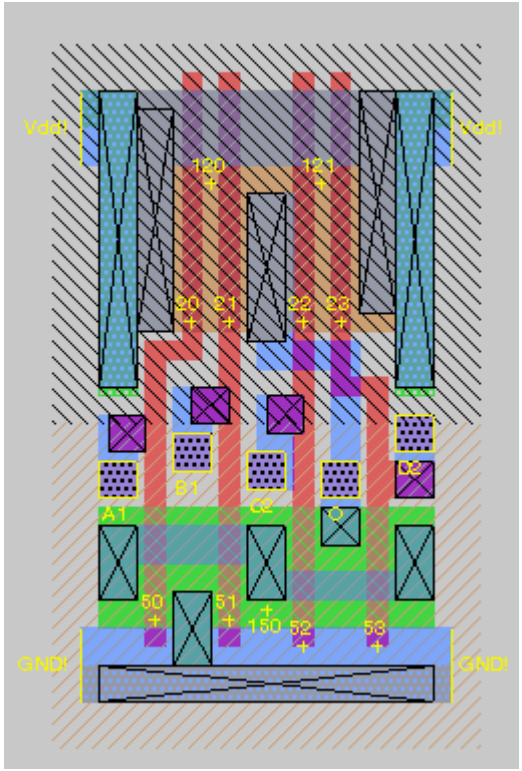
The total of transistors is now 10

OAI: Or-And-Invert

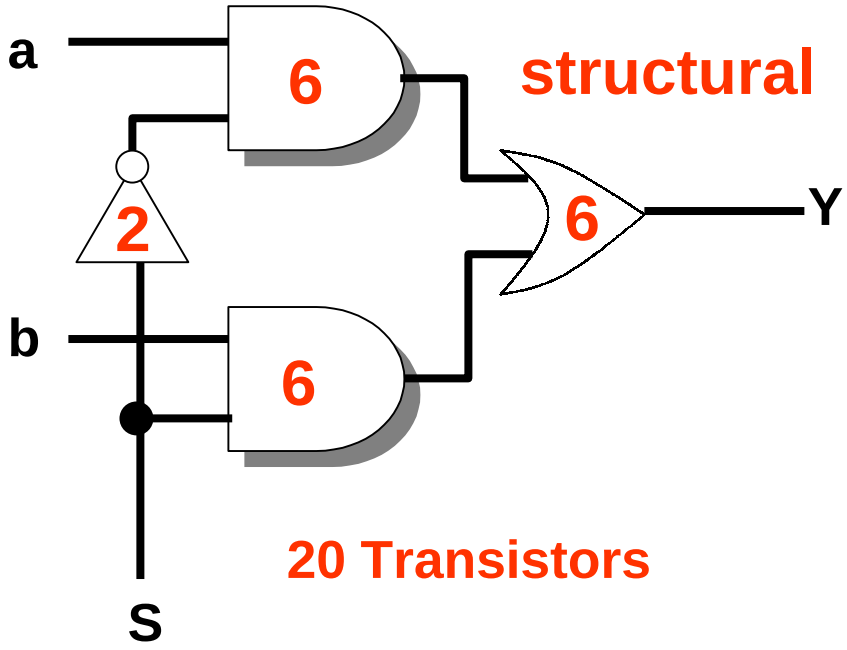
- Or-And-Inverts are dual of the AOIs







with-select-when: 2-to-1 Multiplexor

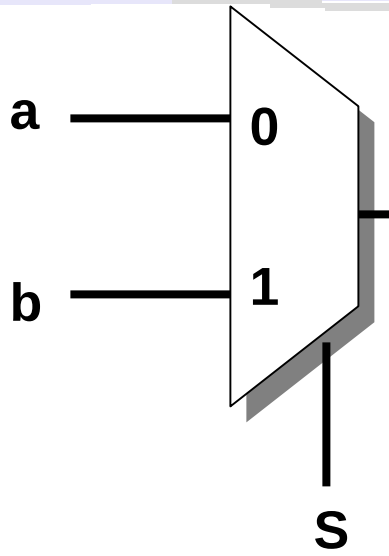


structural

20 Transistors

```
Y <= (a AND NOT s)
      OR
     (b AND s);
```

combinatorial logic



behavioral

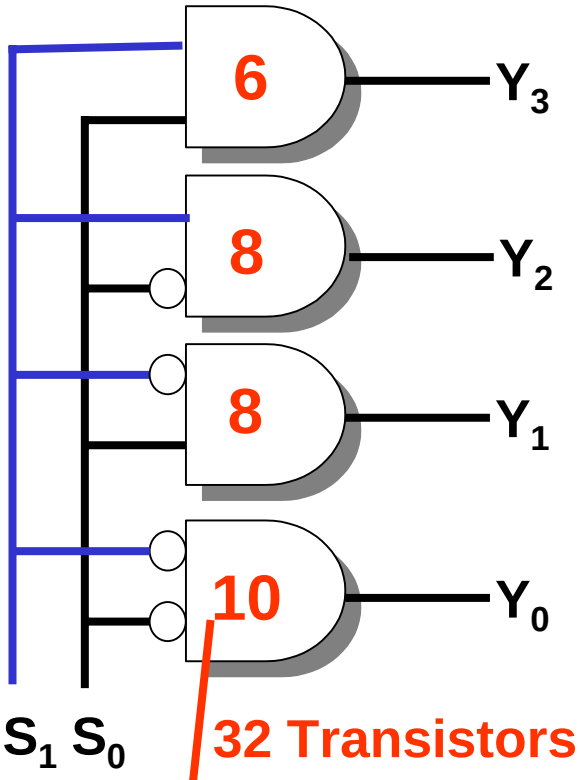
Only values allowed

```
WITH s SELECT
     Y <= a WHEN '0',
          b WHEN '1';
```

or alternatively

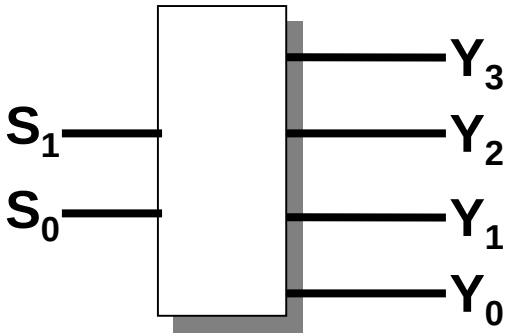
```
WITH s SELECT
     Y <= a WHEN '0',
          b WHEN OTHERS;
```

with-select-when: 2 to 4-line Decoder



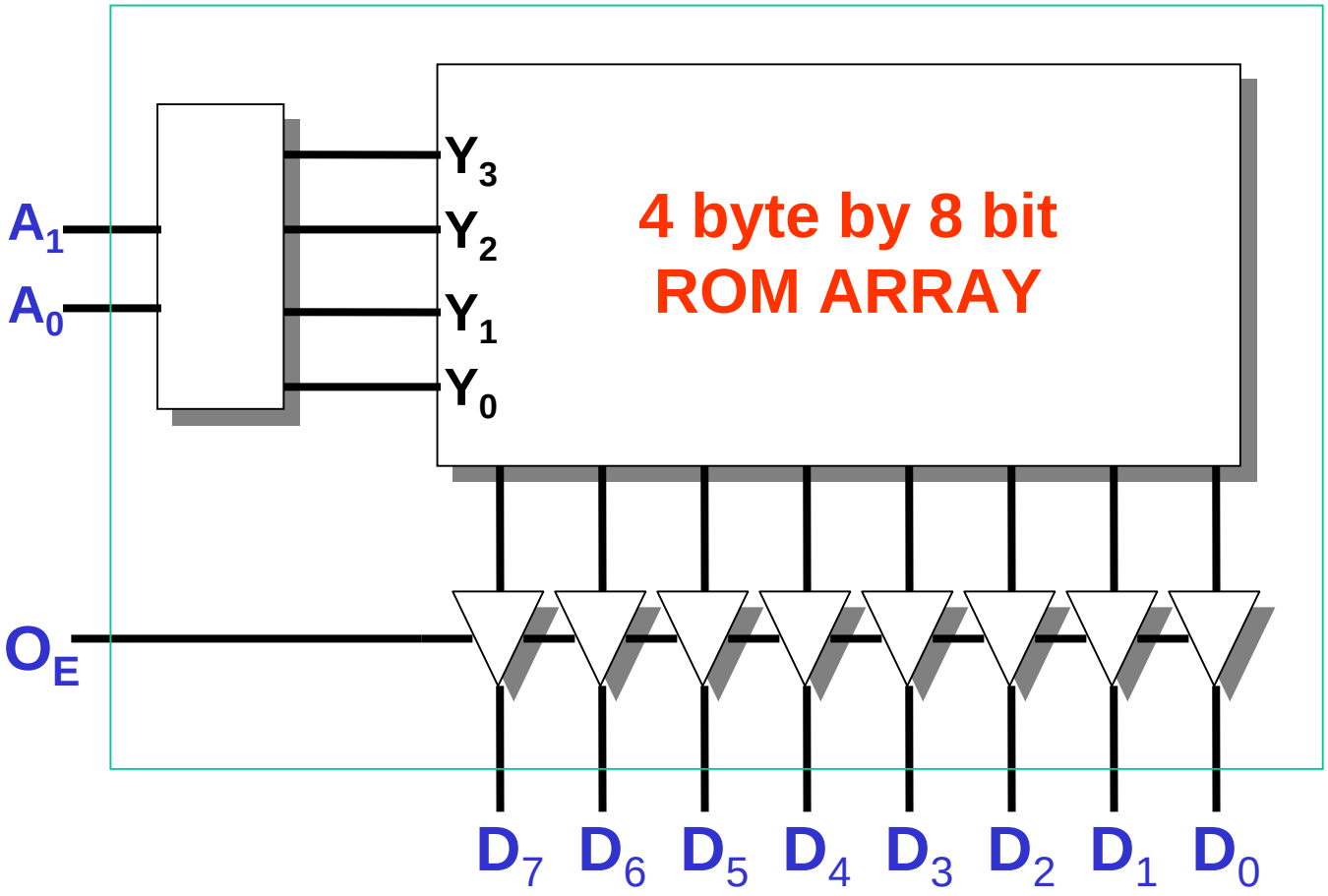
Replace this with a NOR, then 26 total transistors

```
SIGNAL S: std_logic_vector(1 downto 0);
SIGNAL Y: std_logic_vector(3 downto 0);
```




```
WITH S SELECT
    Y <= "1000" WHEN "11",
         "0100" WHEN "10",
         "0010" WHEN "01",
         "0001" WHEN OTHERS;
```

ROM: 4 byte Read Only Memory



ROM: 4 byte Read Only Memory

```
ENTITY rom_4x8 IS
        PORT(A:     IN std_logic_vector(1 downto 0);
                OE:    IN std_logic; -- Tri-State Output
                D:     OUT std_logic_vector(7 downto 0)
); END;
```

```
ARCHITECTURE rom_4x8_arch OF rom_4x8 IS
        SIGNAL ROMout: std_logic_vector(7 downto 0);
BEGIN
  BufferOut: TriStateBuffer GENERIC MAP(8)
                                PORT MAP(D, ROMout, OE);
  WITH A SELECT
        ROMout <=  “01000001” WHEN “00”,
                        “11111011” WHEN “01”,
                        “00000110” WHEN “10”,
                        “00000000” WHEN “11”;
```

when-else: 2-to-1 Multiplexor

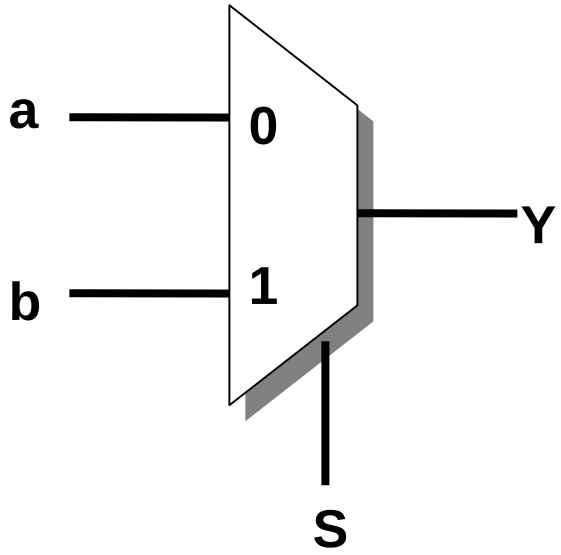
```
WITH s SELECT
   Y <= a WHEN '0',
        b WHEN '1';
```

```
Y <= a WHEN s = '0' ELSE
     b WHEN s = '1';
```

or alternatively

```
WITH s SELECT
   Y <= a WHEN '0',
        b WHEN OTHERS;
```

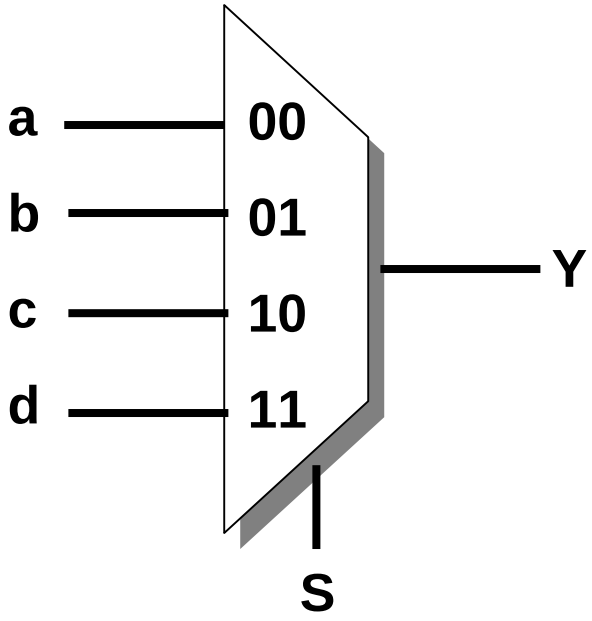
```
Y <= a WHEN s = '0' ELSE
     b;
```



WHEN-ELSE condition allows a condition as part of the WHEN

whereas the WITH-SELECT only allows only a value as part of the WHEN.

with-select-when: 4-to-1 Multiplexor



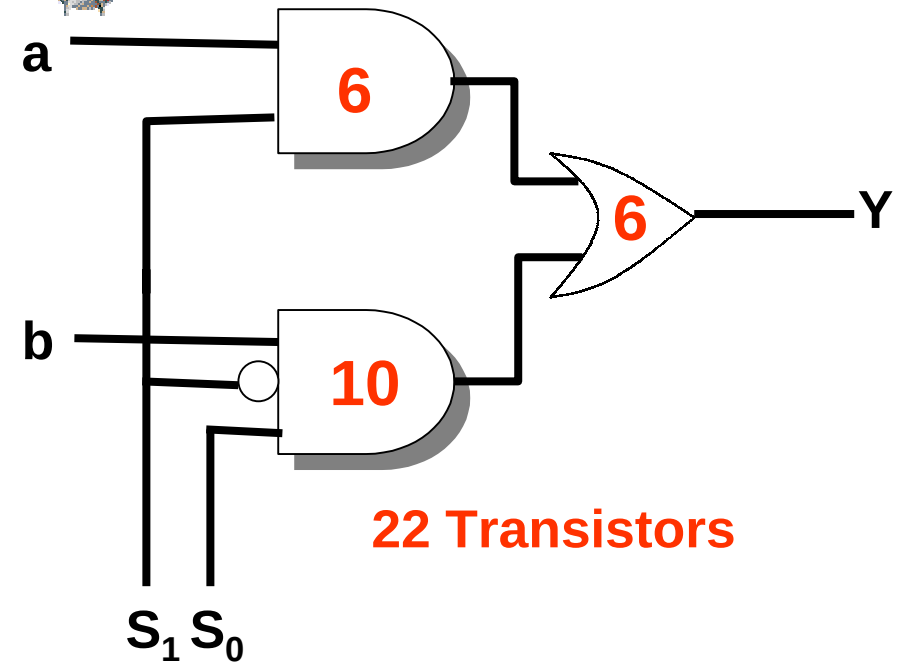

```
WITH s SELECT
  Y <= a WHEN “00”,
       b WHEN “01”,
       c WHEN “10”,
       d WHEN OTHERS;
```

```
Y <=a WHEN s = “00” ELSE
     b WHEN s = “01” ELSE
     c WHEN s = “10” ELSE
     d ;
```

As long as each WHEN-ELSE condition is **mutually exclusive**,

then it is equivalent to the WITH-SELECT statement.

when-else: 2-level priority selector



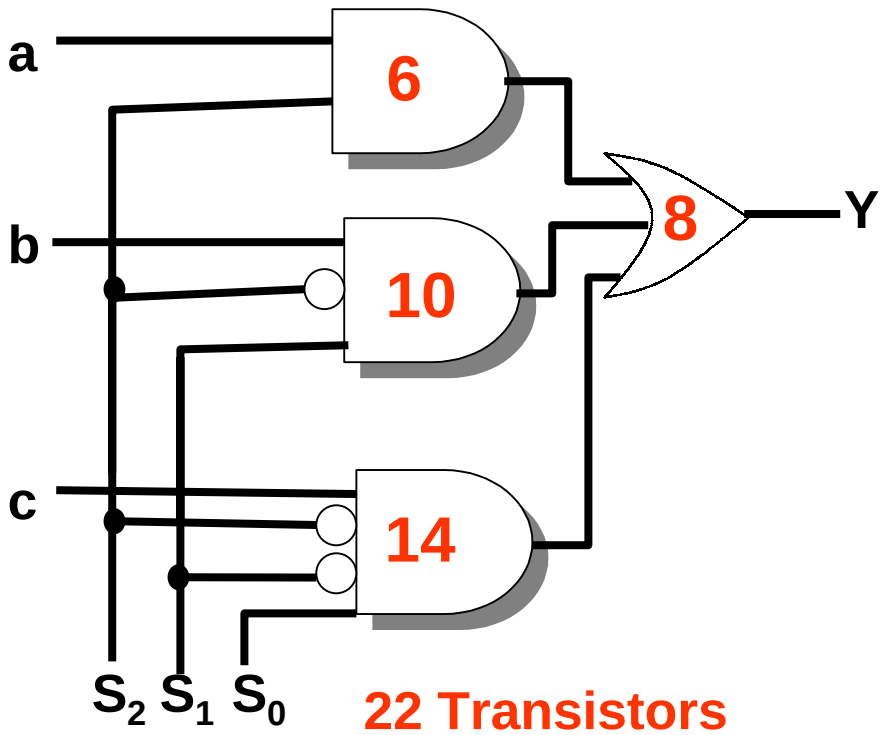

```
Y <= a WHEN s(1) = '1'
ELSE
      b WHEN s(0) = '1'
ELSE
      '0';
```

```
WITH s SELECT
  Y <= a WHEN "11",
       a WHEN "10",
       b WHEN "01",
      '0' WHEN  OTHERS;
```

WHEN-ELSE are useful for sequential or priority encoders

WITH-SELECT-WHEN are useful for parallel or multiplexors

when-else: 3-level priority selector

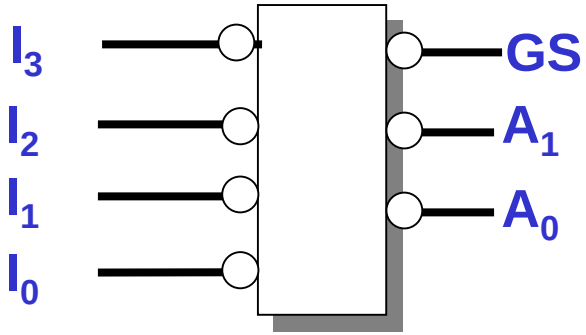



```
WITH s SELECT
  Y <= a WHEN “111”,
       a WHEN “110”,
       a WHEN “101”,
       a WHEN “100”,
       b WHEN “011”,
       b WHEN “010”,
       c WHEN “001”,
      ‘0’ WHEN  OTHERS;
```

```
Y <= a WHEN s(2) = ‘1’ ELSE
     b WHEN s(1) = ‘1’ ELSE
     c WHEN s(0) = ‘1’ ELSE
    ‘0’;
```

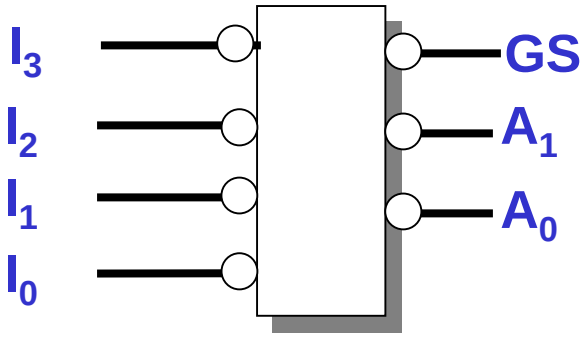
when-else: 2-Bit Priority Encoder (~74LS148)

- Priority encoders are typically used as interrupt controllers
- The example below is based on the 74LS148



I_3	I_2	I_1	I_0	G_S	A_1	A_0
0	X	X	X	0	0	0
1	0	X	X	0	0	1
1	1	0	X	0	1	0
1	1	1	0	0	1	1
1	1	1	1	1	1	1

when-else: 2-Bit Priority Encoder (~74LS148)

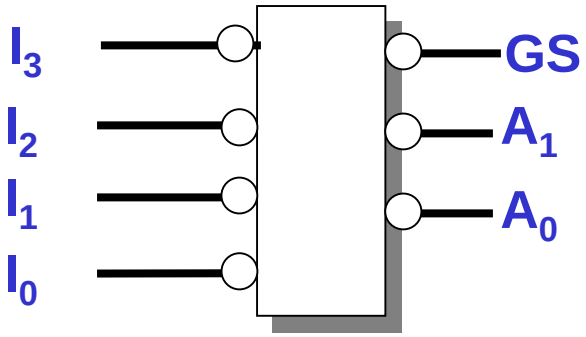



```
ENTITY PriEn2 IS PORT(
I:      IN   std_logic_vector(3 downto 0);
GS:     OUT  std_logic;
A:      OUT  std_logic_vector(1 downto 0);
); END;
```

I_3	I_2	I_1	I_0	G_S	A_1	A_0
0	X	X	X	0	0	0
1	0	X	X	0	0	1
1	1	0	X	0	1	0
1	1	1	0	0	1	1
1	1	1	1	1	1	1

```
A <= "00" WHEN I3 = 0 ELSE
     "01" WHEN I2 = 0 ELSE
     "10" WHEN I1 = 0 ELSE
     "11" WHEN I0 = 0 ELSE
     "11" WHEN OTHERS;
```

when-else: 2-Bit Priority Encoder (~74LS148)



I_3	I_2	I_1	I_0	G_S	A_1	A_0
0	X	X	X	0	0	0
1	0	X	X	0	0	1
1	1	0	X	0	1	0
1	1	1	0	0	1	1
1	1	1	1	1	1	1

Structural model

```
GS <= NOT( NOT(I3) OR NOT(I2)
        OR NOT(I1) OR NOT(I0) )
```

Structural model

```
GS <= I3 AND I2 AND I1 AND I0
```

Behavioral model

```
GS <= WITH I SELECT
        '1' WHEN "1111",
        '0' WHEN OTHERS;
```