A Cubical technique for Minimization

- The initial specification of the function is given in the form of implicants (not necessarily prime or minterms.
- Use the intersection ($*$ operation) as defined earlier
- First, find all the prime implicants
- Find the essential prime implicants
- Find minimum coverage

A Cubical technique for Minimization

- Let $A=\{A_1, A_2, \ldots A_n\}, B=\{B_1, B_2, \ldots B_n\}$ be two implicants
- Find $C=A * B$
  - C= ϕ if $A_i * B_i = \phi$ for more than one *i*
  - $c_i=A_i * B_i$ *where* $A_i * B_i \neq \phi$, and c_i = x if $A_i * B_i = \phi$

$\cap$	0	1	X
0	0	ϕ	0
1	ϕ	1	1
X	0	1	x

A Cubical technique for Minimization cont.

- Find new cubes that are not included in the existing cubes.
- $G^{k+1} = c_i * c_j$ for all c_i, c_j in C^k
- $C^{k+1} = C^k \cup G^{k+1}$ - redundant cubes
- Repeat until $C^{k+1} = C^k$, then we have the set of all prime implicants

Example

Finding Essential Prime Implicants

Define the # operation

operation returns the part of A that is not covered by B

C = A # B such that

C = A if A_i # B_i = ϕ for some i

C = ϕ if A_i # B_i = ε for all i

Otherwise C = $\cup_i(A_1, A_2, \ldots B_i', ..A_n)$ where the union is for all I for which A_i=x and $B_i \neq$ x.

	0	1	x
0	ε	ϕ	ε
1	ϕ	ε	ε
X	1	0	ε

A_i # B_i

Finding Essentials Prime Implicants

- Lep P be the set of all prime implicants, and DC is the set of all don't care.
- Let pi denote one prime implicant
- Pi is an essential prime implicant if

$$p^i \# (P - p^i) \# DC \neq \phi$$

- The meaning of is that the # operation is applied successively to each prime implicant in in P, if P={p^1,p^2,p^3,p^4) and DC={d^1,d^2} then we evaluate

$$((((p^3 \# p^1) \# p^2) \# p^4) \# d^1) \# d^2$$

Finding Essentials Prime Implicants

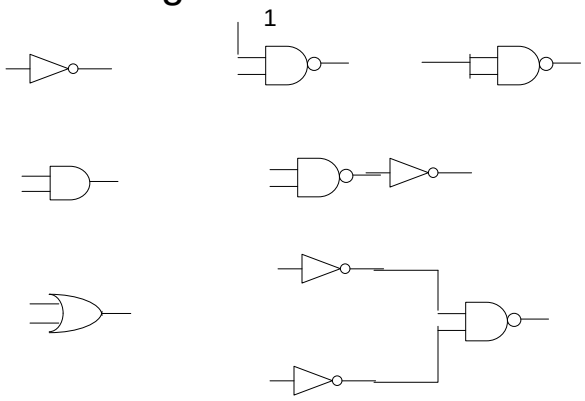
- Example
- Consider the set {x01x, x101, 01x1, xx10)

Complete Procedure

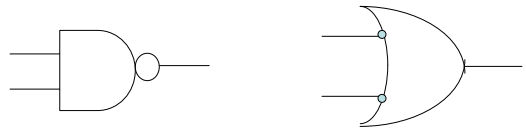
- Let C^0 = set of initial cover of *f* and DC
- Find all prime implicants of C^0 = P
- Find the essential prime implicants. If that set covers all the vertices of *f* this is the minimum cover
- Delete any non essential p^i that is more expensive than other prime implicant p^j if p^i # DC # p^j = ϕ
- Choose the lowest-cost prime implicants to cover the remaining vertices of *f*. Use branching

NAD and NOR Implementation

- Universal gate



NAND Gate



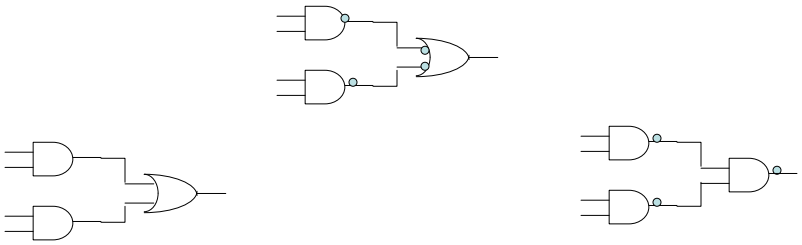
AND-invert

Invert-OR

2 graphic symbols for NAND gate

2-Level Implementation

- Start with sum of product F=AB+CD



NAND Implementation

- Express the function in sum of products
- Replace every AND by a NAND
- Replace the OR by Invert-OR
- If a single element is an input to the OR invert it.
- Change invert-OR to AND-invert
- Example on 3 variables

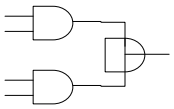
Multilevel NAND Circuits

- Convert all AND gates to NAND gates with AND-invert symbols
- Convert all OR gates to NAND gates with invert-OR symbols
- Check all the bubbles in the diagram, for every bubble that is not compensated by another bubble on the same line, add an inverter (or compliment the input literal)

NOR Implementation

Wired Logic

- Wired AND in open collector TTL
- Wired-OR in ECL gates

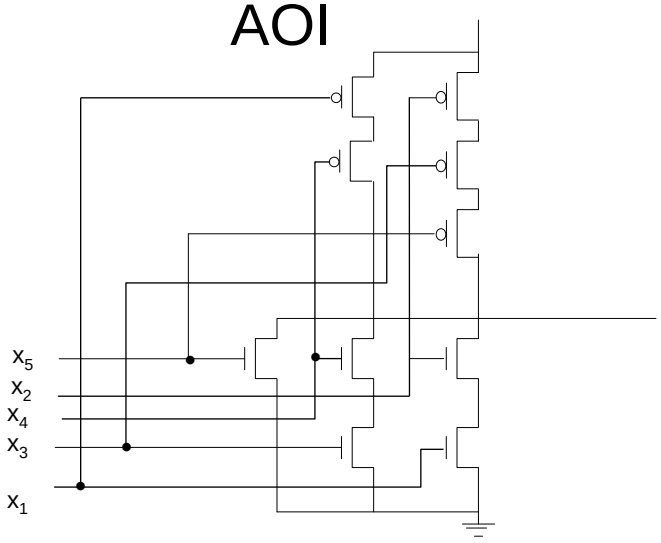


Wired-AND in TTL

AND-OR-INVERT AOI

- In CMOS, and in most other logic families, the simplest gates are inverters, then NAND and NOR gates.
- It is typically not possible to design a noninverting gate with less transistors than an inverting gate.
- CMOS circuits can perfrom two level of logic with just a single level of transistors. (AOI gate).
- The speed an other electrical characteristics of a CMOS AOI or OAI gate is quite comparable to those of a single CMOS NAND or NOR.

AOI

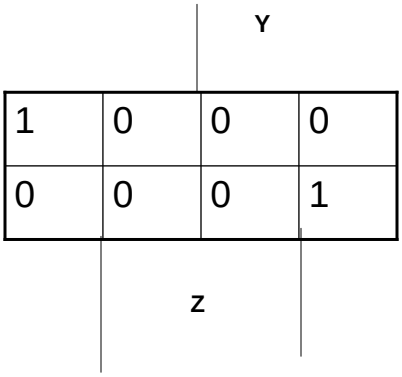


AOI Gates

- If you implement the complement of the function in sum of products it results in AOI circuit

Other 2-level Implementation

- Consider the function F=x'y'z'+zyz'
- Take the complement F'=x'y+xy'+z
- You can implement it as AOI using F'
- Change it to NAND-AND by moving the bubble from the output of the OR to its inputs (and changing it to AND) NAND-AND implementation

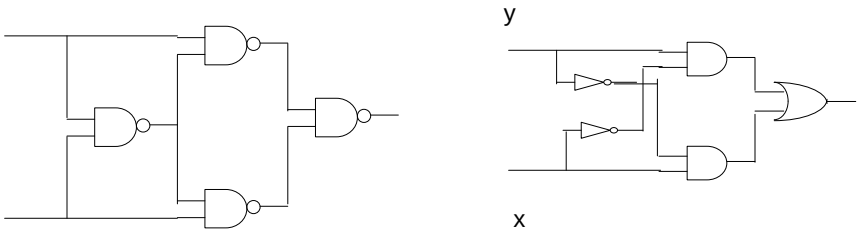


Other 2-level Implementation

- Using product of sums
- F=z'(x+y')(x'+y), OR we can say
- F'=[(x'+y'+z)(x+y+z)]
- We can implement the above equation using OR and NAND (OR-NAND implementation).
- Then we can move the bubble of the NAND to its inputs and changing it to OR (NOR-OR implementation)

EX-OR

- $x \oplus y = x'y + y'x$



- 3 and 4 inout EX-OR Parity

HDL

- Can be used to represent logic diagram, Boolean expressions, and finite state machine representation.
- Used to document digital systems
- Used in simulation and synthesis
- 2 main languages, Verilog, and VHDL

Verilog

- C-like syntax
- Case sensitive, // for comments

```
module smpl_circuit(A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and g1(e,A,B);
   not g2(y,c);
   or g3(x,e,y);
endmodule
```

Verilog

- We can introduce delay

```
module smpl_circuit(A,B,C,x,y);
   input A,B,C;
   output x,y;
   wire e;
   and #(30) g1(e,A,B);
   not #(20) g2(y,c);
   or #(10) g3(x,e,y);
endmodule
```


Verilog

```
// Behavioral Model of a Nand gate
// By Dan Hyde, August 9, 1995
module NAND(in1, in2, out);
input in1, in2;
output out;
// continuous assign statement
assign out = ~(in1 & in2);
endmodule
```

- The continuous assignment statement is used to model **combinational circuits** where the outputs change when one wiggles the input.

Verilog

```
module AND(in1, in2, out);
// Structural model of AND gate from two NANDS
input in1, in2;
output out;
wire w1;
// two instantiations of the module NAND
NAND NAND1(in1, in2, w1);
NAND NAND2(w1, w1, out);
endmodule
```


Testing

```
module test_AND;
// High level module to test the two other modules
reg a, b;
wire x,y;
Circuit_with_delay cwd(A,B,C,x,y);
initial
    begin  // Test data
    A=1'b0; B=1'b0; C=1'b0;
    #100  A=1'b1;  B=1'b1;  C=1'b1;
    #100 $finish;
    end
endmodule
Module circuit_with_delay(A,B,C,x,y);
     input A,B,C;
     output x,y;
     wire e;
     and #(30) g1(e,A,B);
     not #(20) g2(y,c);
     or #(10) g3(x,e,y);
endmodule
```

User Defined Primitives

```
//User defined primitive(UDP)
primitive crctp(x,A,B,C);
    output x;
    input A,B,C;
//Now the truth table
    table
//  A     B         C         :         x
    0     0         0         :         1;
    0     0         1         :         0;
    0     1         0         :         1;
    0     1         1         :         0;
    1     0         0         :         1;
    1     0         1         :         0;
    1     1         0         :         1;
    1     1         1         :         1;
    endtable
Endprimitive
module abcdef;
    reg x,y,z;
    wire w;
    crctp(w,x,y,z);
endmodule
```