

Diseño de Sistemas Digitales

Cap2 parte2

FCHE 2011-2

2 Circuitos combinacionales

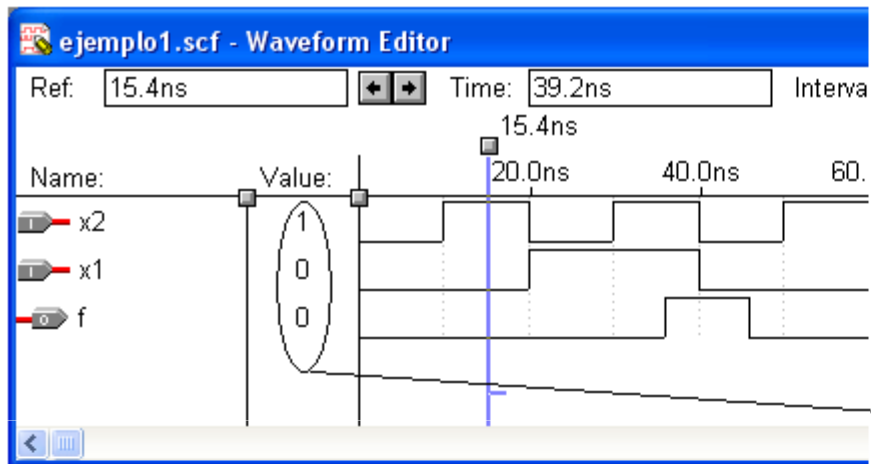
Objetivo: El alumno conocerá los componentes electrónicos básicos involucrados en los circuitos combinacionales, así como los bloques funcionales combinacionales más utilizados en el diseño de sistemas digitales tanto en descripción estructural como por comportamiento usando HDL.

Contenido:

- 2.1 Compuertas lógicas AND, OR, NOT, NAND, NOR, XOR, XNOR
- 2.2 Formas canónicas, estándar, mintérminos y maxtérminos
- 2.3 Minimización de funciones booleanas con mapas de Karnaugh y Quine-McCluskey
- 2.4 Circuitos integrados, familias lógicas
- 2.5 Interpretación de parámetros en las hojas de datos de las compuertas lógicas
- 2.6 Convenciones de lógica positiva y lógica negativa
- 2.7 Representación estructural y por comportamiento de las compuertas lógicas en algún lenguaje de descripción de hardware (HDL)
- 2.8 Implementación estructural y por comportamiento de Funciones Booleanas usando algún HDL
- 2.9 Universalidad de las compuertas NAND y NOR
- 2.10 Propiedades de la XOR y NXOR para la implementación de generadores y detectores de paridad
- 2.11 Análisis de tiempo en la implementación de funciones booleanas
- 2.12 Descripción estructural y por comportamiento, usando HDL, de los bloques combinacionales fundamentales: Medio sumador, sumador completo, sumador/restador de "n" bits, comparadores de "n" bits, sumadores BCD, multiplicadores de "NxM" bits, decodificadores, codificadores, decodificadores BCD – 7 SEG, multiplexores, demultiplexores
- 2.13 Dispositivos lógicos programables elementales: ROM, PLA, PAL
- 2.14 Implementación de funciones booleanas con decodificadores, multiplexores, ROM, PLA y PAL

VHDL/ Análisis de tiempo

Compuertas Lógicas: AND

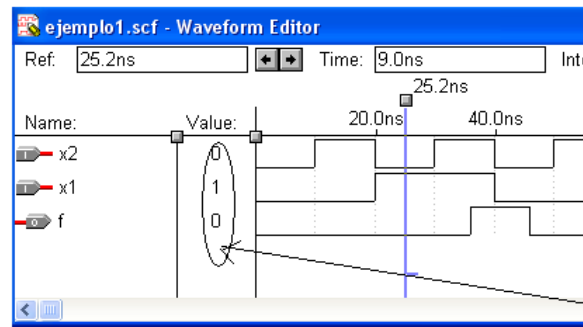


```

class1.vhd - Text Editor
ENTITY ejemplo1 IS
    PORT ( x1, x2 : IN    BIT ;
          f  : OUT   BIT ) ;
END ejemplo1 ;

ARCHITECTURE LogicFunc OF ejemplo1 IS
BEGIN
    f <= x1 AND x2;
END LogicFunc ;
    
```

AND		
x1	x2	f
0	0	0
0	1	0
1	0	0
1	1	1



```

class1.vhd - Text Editor
ENTITY ejemplo1 IS
    PORT ( x1, x2 : IN    BIT ;
          f  : OUT   BIT ) ;
END ejemplo1 ;

ARCHITECTURE LogicFunc OF ejemplo1 IS
BEGIN
    f <= x1 AND x2;
END LogicFunc ;
    
```

AND		
x1	x2	f
0	0	0
0	1	0
1	0	0
1	1	1

Tarea: instalar Altera y correr estos programas, subir relaciones. Checar pdf tutorialvhdl

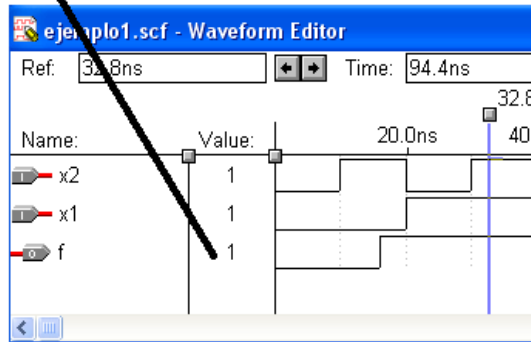
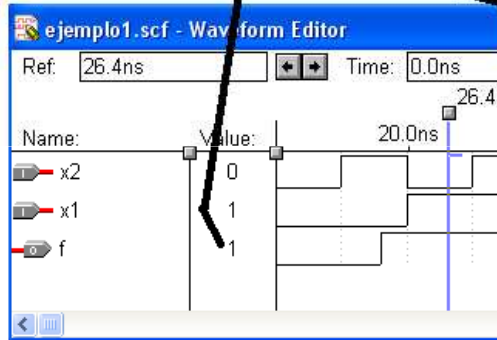
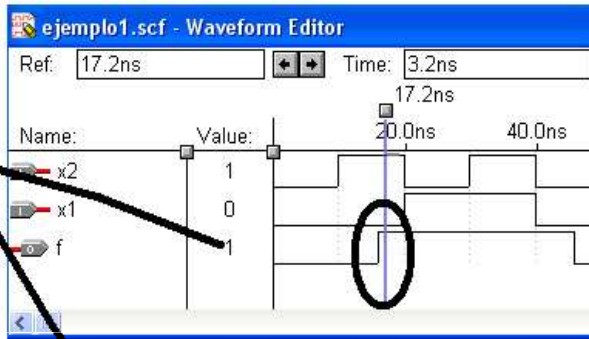
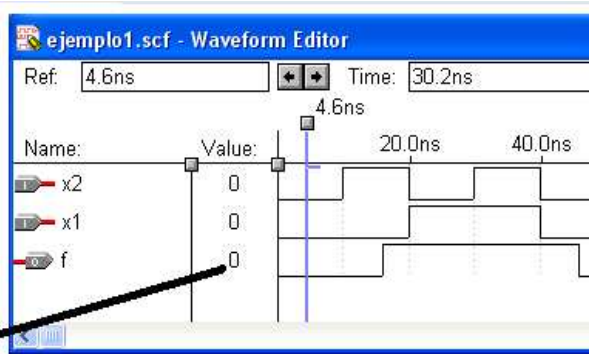
```

clase1.vhd - Text Editor
ENTITY ejemplo1 IS
  PORT ( x1, x2 : IN  BIT ;
        f : OUT  BIT ) ;
END ejemplo1 ;

ARCHITECTURE LogicFunc OF ejemplo1 IS
BEGIN
  f <= x1 OR x2;
END LogicFunc ;

```

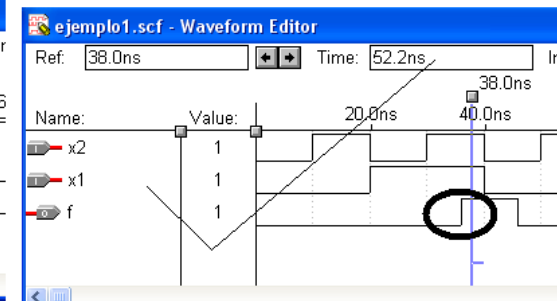
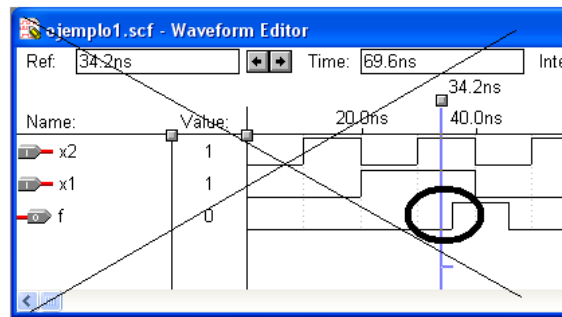
OR		
x1	x2	f
0	0	0
0	1	1
1	0	1
1	1	1



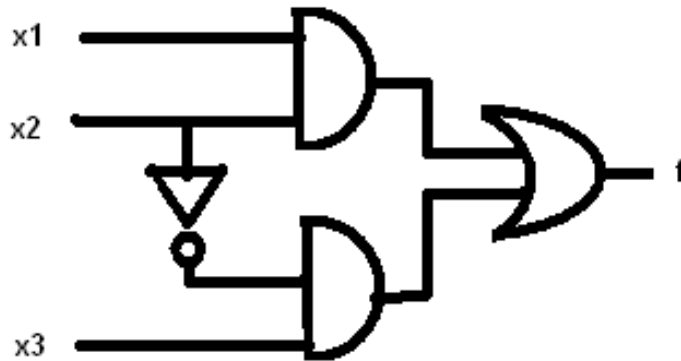
Compuertas Lógicas: OR

Relaciones entre tabla de verdad, programa y análisis de tiempo

Errores que deben evitarse

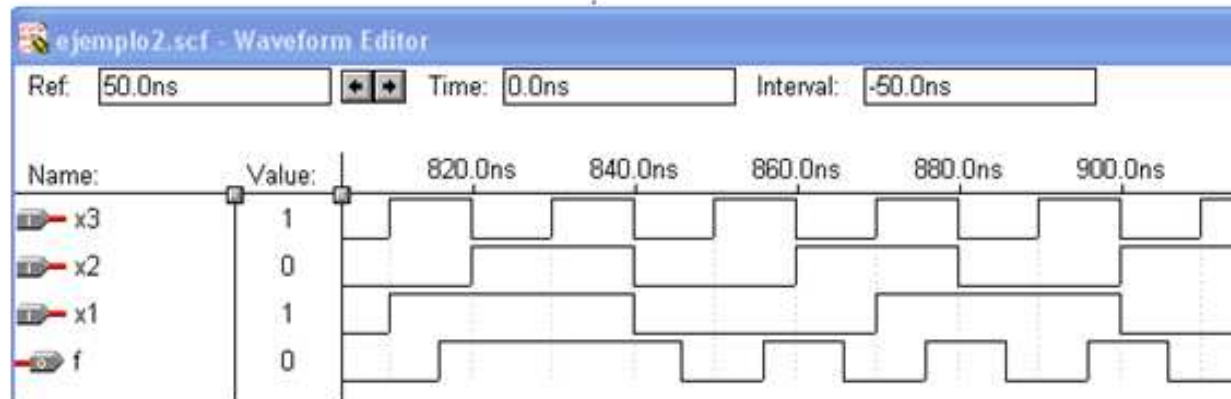


VHDL con funciones Booleanas



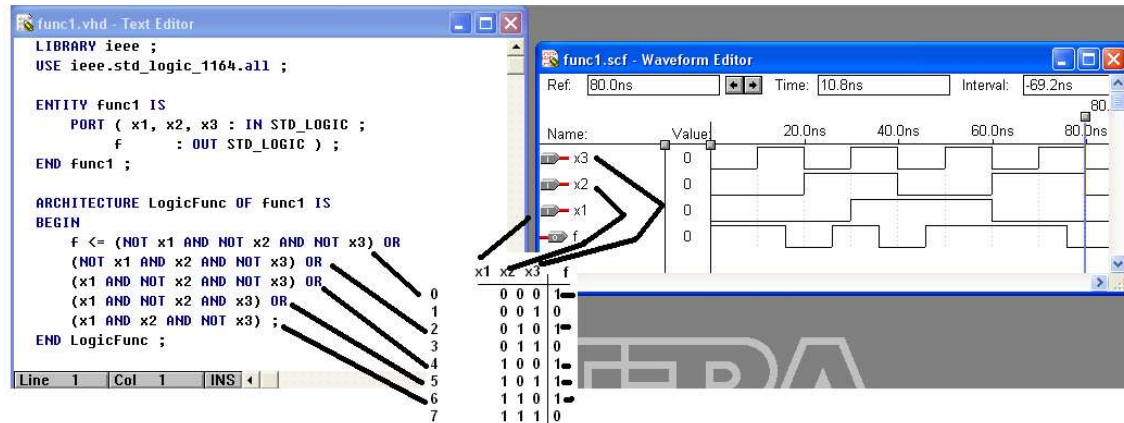
```
Untitled3 - Text Editor
ENTITY ejemplo2 IS
    PORT(
        x1,x2,x3 : IN bit;
        f       : OUT bit);
END ejemplo2;

ARCHITECTURE simplificacion1 OF ejemplo2 IS
    BEGIN
        f<=(x1 and x2) or (not x2 and x3);
    END simplificacion1;
```



con tipos de datos BIT. Por lo tanto su comportamiento se escribirá con respecto a estos bits. Debimos haber sintetizado antes. SIN TABLA DE VERDAD. SOLO CON FUNCION

VHDL con tablas de verdad



func1.vhd - Text Editor

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
```

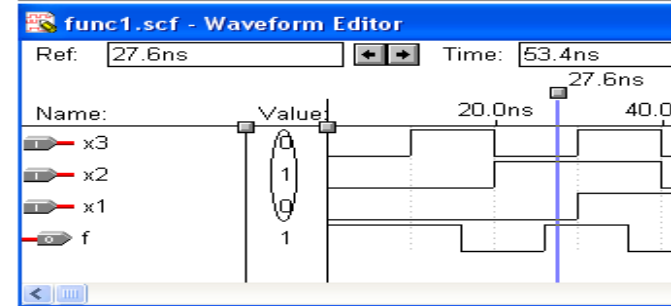
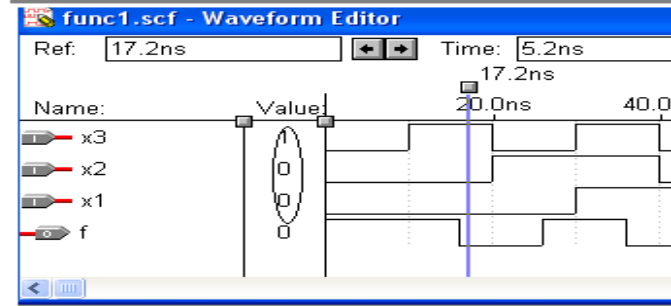
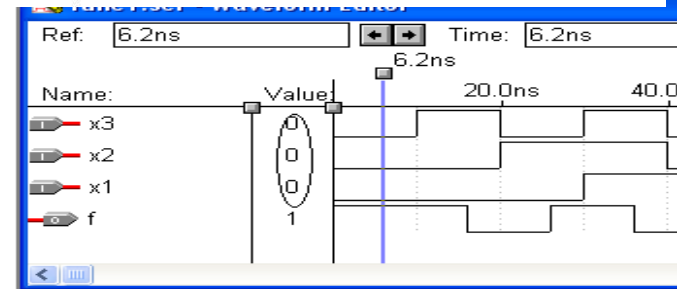
```
ENTITY func1 IS
    PORT ( x1, x2, x3 : IN STD_LOGIC ;
          f      : OUT STD_LOGIC ) ;
END func1 ;
```

```
ARCHITECTURE LogicFunc OF func1 IS
BEGIN
    f <= (NOT x1 AND NOT x2 AND NOT x3) OR
        (NOT x1 AND x2 AND NOT x3) OR
        (x1 AND NOT x2 AND NOT x3) OR
        (x1 AND NOT x2 AND x3) OR
        (x1 AND x2 AND NOT x3) ;
END LogicFunc ;
```

x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

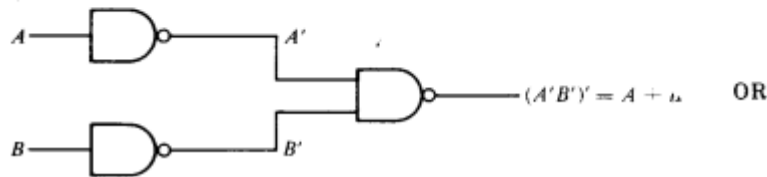


Sin reducir la función, directamente de la tabla. El tipo de dato STD_LOGIC requiere cabecera de librería.

2.9 Universalidad Nand Nor

2.10 XOR XNOR gen. Y detect.
paridad

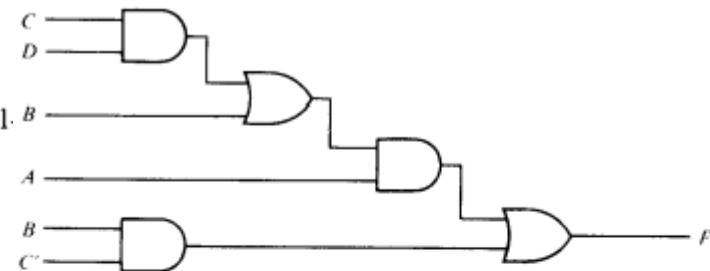
La compuerta NAND se conoce como la compuerta universal ya que cualquier sistema digital se puede configurar con ella.



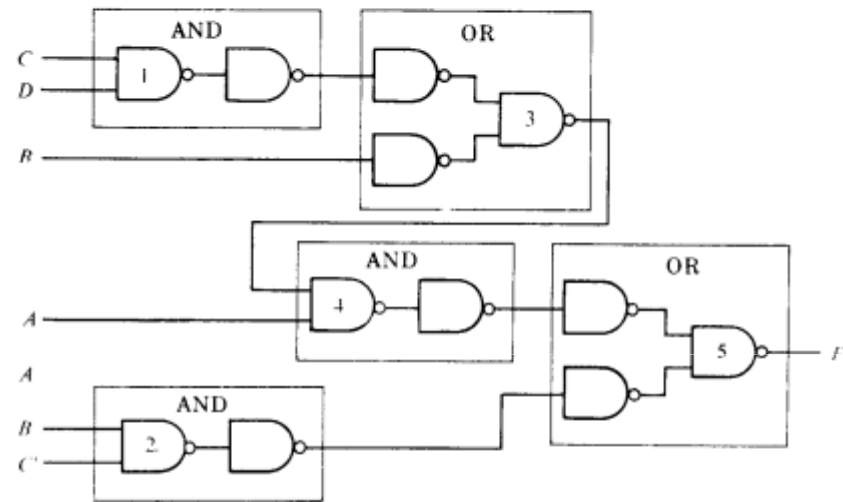
1. A partir de una expresión algebraica, dibújese el diagrama lógico con compuertas AND, OR y NOT. Asígnese que se tienen disponibles las entradas normales y sus compuertas.
2. Dibújese un segundo diagrama lógico con la lógica NAND equivalente, como se da en la Figura 4-10 y sustitúyase para cada compuerta AND, OR y NOT.
3. Quítense cualquier par de inversores en cascada del diagrama ya que la doble inversión no produce una función lógica. Quítense los inversores conectados a entradas externas simples y complementétese la variable de entrada correspondiente. El nuevo diagrama lógico obtenido es la configuración con compuertas NAND requerido.

Este procedimiento se ilustra en la Figura 4-11 para la función:

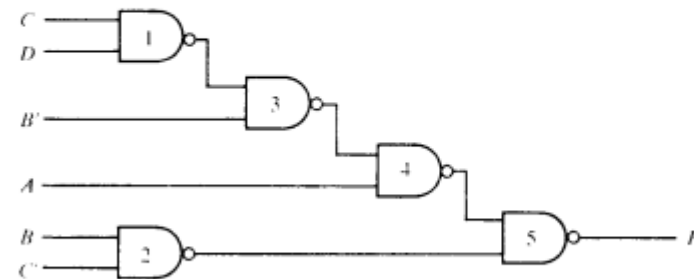
$$F = A(B + CD) + BC'$$



(a) Configuración AND-OR



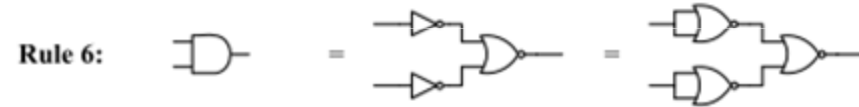
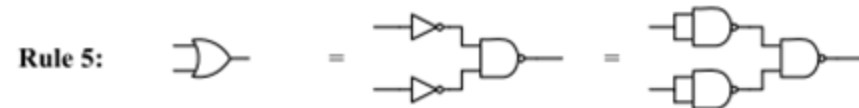
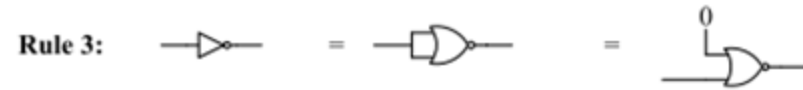
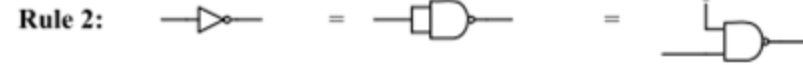
(b) Sustituyendo funciones NAND equivalentes de la Figura 5-8



(c) Configuración con NAND

Figura 4-11 Configuración de $F = A(B + CD) + BC'$ con compuertas NAND

Convirtiendo a NAND y NOR



Rule 1: $x'' = x$ (double NOT)

Rule 2: $x' = (x \bullet x)' = (x \bullet 1)'$ (NOT to NAND)

Rule 3: $x' = (x + x)' = (x + 0)'$ (NOT to NOR)

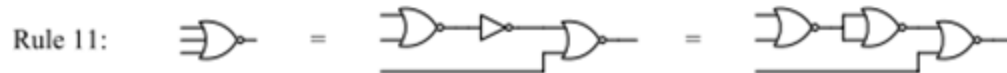
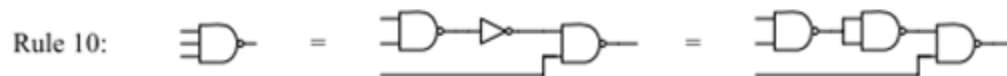
Rule 4: $xy = ((xy)')'$ (AND to NAND)

Rule 5: $x + y = ((x + y)')' = (x' y')'$ (OR to NAND)

Rule 6: $xy = ((xy)')' = (x' + y')'$ (AND to NOR)

Rule 7: $x + y = ((x + y)')'$ (OR to NOR)

Figure 3.4 Circuits for converting from AND, OR, and NOT gates to NAND or NOR gates.



Rule 8: $(x \bullet y)' = (x \bullet y \bullet 1)'$ (2-input to 3-input NAND)

Rule 9: $(x + y)' = (x + y + 0)'$ (2-input to 3-input NOR)

Rule 10: $(abc)' = ((ab) c)' = ((ab)' c)'$ (3-input to 2-input NAND)

Rule 11: $(a+b+c)' = ((a+b) + c)' = ((a+b)'' + c)'$ (3-input to 2-input NOR)

OR Exclusiva

Las funciones de OR-exclusiva y de equivalencia son muy útiles en sistemas que requieren códigos de detección y corrección de errores.

un bit de paridad es una forma de detectar errores durante la transmisión de información binaria.

Un bit de paridad

es un bit extra incluido con un mensaje binario para hacer el número de unos par o impar.

El mensaje, incluyendo el bit de paridad, se transmite y luego se comprueba en el extremo de recepción los errores.

Un error se

detecta si la paridad comprobada no corresponde a la transmitida.

El circuito que genera el bit de paridad en un transmisor se llama *generador de paridad*; el circuito que comprueba la paridad en el receptor se llama *comprobador de paridad*.

Tabla 4-4 Generación de paridad impar.

Mensaje de tres bits			Bit de paridad generado
x	y	z	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

En m0 no tenemos paridad, le agregamos un UNO y ya tiene paridad impar.

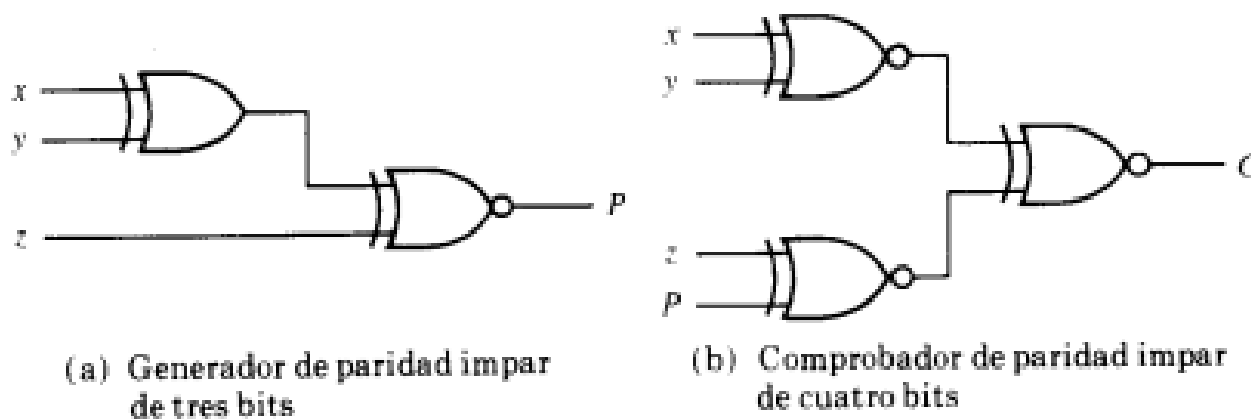


Figura 4-24 Diagramas lógicos para la generación y comprobación de la paridad