

Tema 1: Sist. Digitales: Circuitos Combinacionales

Enero de 2011

1 Introducción

- Álgebra de Boole. Funciones lógicas
- Mapas de Karnaugh

2 Circuitos Combinacionales Comunes

- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- Bloques lógicos
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Agenda

1 Introducción

- Álgebra de Boole. Funciones lógicas
- Mapas de Karnaugh

2 Circuitos Combinacionales Comunes

- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- Bloques lógicos
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Introducción

- **Ordenador:** Dispositivo digital → Información representada de forma discreta, en lugar de continua.
- **Electrónica digital:** Dos niveles de tensión (alta/baja = 1/0) → Uso del sistema de numeración binario como abstracción de dichos estados.
- **Circuitos combinacionales (sin memoria):** Las salidas dependen sólo de las entradas actuales. (Objeto de estudio en este tema)
- **Circuitos secuenciales (con memoria):** Las salidas dependen de las entradas y del valor almacenado en su memoria (estado). (Objeto de estudio en el tema 2)

Agenda

1 Introducción

- Álgebra de Boole. Funciones lógicas
- Mapas de Karnaugh

2 Circuitos Combinacionales Comunes

- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- Bloques lógicos
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Álgebra de Boole

- El Álgebra de Boole es muy adecuada para expresar y analizar circuitos lógicos. Utilizaremos la Álgebra de Boole formada por $\{0,1\}, +, \cdot$; donde los operadores son la suma y el producto lógico
- Sobre ella se pueden definir funciones de n variables $F : \{0, 1\}^n \rightarrow \{0, 1\}$ que puede describirse mediante:
 - Ecuación lógica:

$$F(A, B, C) = \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

- Tabla de verdad:

| ENTRADAS | | | SALIDA |
|----------|---|---|--------|
| A | B | C | F |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Formas Canónicas. Minitérminos y Maxitérminos

- Una misma función lógica puede expresarse mediante infinitas ecuaciones lógicas. Nos centraremos en las formas normalizadas:
 - Suma de productos (minitérminos):

$$F(A, B, C) = \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + A \cdot B \cdot \bar{C} + A \cdot B \cdot C$$

$$F(A, B, C) = m_2 + m_3 + m_6 + m_7 = \sum m(2, 3, 6, 7)$$

- Producto de sumas (maxitérminos):

$$F(A, B, C) = (A+B+C) \cdot (A+B+\bar{C}) \cdot (\bar{A}+B+C) \cdot (\bar{A}+B+\bar{C})$$

$$F(A, B, C) = M_0 \cdot M_1 \cdot M_4 \cdot M_5 = \prod M(0, 1, 4, 5)$$

Agenda

1 Introducción

- Álgebra de Boole. Funciones lógicas
- Mapas de Karnaugh

2 Circuitos Combinacionales Comunes

- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- Bloques lógicos
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Mapas de Karnaugh

- Método sencillo para minimizar funciones lógicas, limitado en la práctica hasta 5 ó 6 variables.
- Mapa de Karnaugh = Representación gráfica de una tabla de verdad. Una celda por cada fila de la tabla, mintérminos adyacentes ocupan celdas adyacentes (incluidas adyacencias en los extremos).

| | | A | |
|---|---|---|---|
| | | 0 | 1 |
| B | 0 | | |
| | 1 | | |

| | | AB | | | |
|---|---|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| C | 0 | | | | |
| | 1 | | | | |

| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | | | | |
| | 01 | | | | |
| | 11 | | | | |
| | 10 | | | | |

Criterios de simplificación con Mapas de Karnaugh

- 1 1 cuadrado tiene n cuadrados adyacentes (1 por variable).
- 2 Los cuadrados se combinan en grupos de potencias de 2. Al agrupar 2^k celdas, se eliminan k variables.
- 3 A mayor grupo, menor número de variables en el producto obtenido (puertas AND resultantes con menos entradas).
- 4 Hay que intentar cubrir todos los unos con el menor número de grupos posibles (puerta OR resultante con menos entradas).
- 5 Conviene comenzar por los unos más aislados en el mapa (puesto que los otros ofrecen más posibilidades de combinación).

Simplificación con Mapas de Karnaugh

- Simplificar la función

$$F(A, B, C, D) = \sum m(4, 5, 6, 7, 8, 10, 11, 12):$$

| | | AB | | | |
|----|----|--------|--------|---------|---------|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 0 0 | 1 4 | 1 12 | 1 8 |
| | 01 | 0 1 | 1 5 | 0 13 | 0 9 |
| | 11 | 0 3 | 1 7 | 0 15 | 1 11 |
| | 10 | 0 2 | 1 6 | 0 14 | 1 10 |

- La expresión simplificada de la función queda:

$$F(A, B, C, D) = \bar{A} \cdot B + A\bar{C} \cdot \bar{D} + A \cdot \bar{B} \cdot C$$

Terminología en la simplificación con Karnaugh

- **Implicante:** Producto de variables cualquiera. $\overline{A}\overline{B}\overline{C} = \{4, 5\}$.
- **Implicante primo:** Implicante no contenido en otro.
 $\overline{B}\overline{C}\overline{D} = \{4, 12\}$.
- **Implicante primo esencial:** Implicante primo con al menos un 1 sólo cubierto por él. $\overline{A}B = \{4, 5, 6, 7\}$.
- **Cubierta:** Conjunto de implicantes primos que cubren todos los unos. (debe incluir, al menos, todos los IP esenciales).
 $\overline{A}B + A\overline{C}\overline{D} + A\overline{B}C = \{\{4, 5, 6, 7\}, \{12, 8\}, \{11, 10\}\}$.

| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 0 | 1 | 1 | 1 |
| | 01 | 0 | 1 | 0 | 0 |
| | 11 | 0 | 1 | 0 | 1 |
| | 10 | 0 | 1 | 0 | 1 |

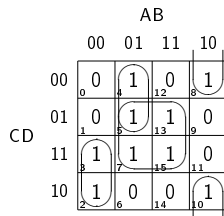
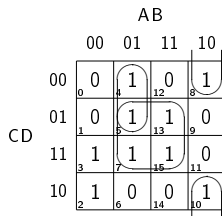
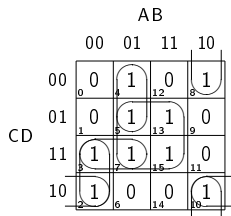
Algoritmo de Minimización con Karnaugh

- 1 Identificar los implicantes primos. Para esto se busca obtener los grupos con mayor cantidad de unos adyacentes. Los grupos deben contener un número de unos que son potencias de 2.
- 2 Identificar todos los implicantes primos esenciales.
- 3 La expresión mínima se obtiene seleccionando todos los implicantes primos esenciales y el menor número de implicantes primos para cubrir los minterminos no incluidos en los implicantes primos esenciales.
⇒ Es en forma de suma de productos.

Ejemplo de Simplificación

Utilizar el mapa de Karnaugh para simplificar la función:

$$F(A, B, C, D) = \sum m(2, 3, 4, 5, 7, 8, 10, 13, 15)$$



Se muestran los implicantes primos, implicantes primos esenciales y cubierta de la función. El resultado es:

$$F(A, B, C, D) = \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + B \cdot D + A \cdot \bar{B} \cdot \bar{D}$$

Ejemplo de Simplificación por ceros

Simplificar *por ceros* la función $\sum m(2, 3, 4, 5, 7, 8, 10, 13, 15)$

| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 0 | 1 | 0 | 1 |
| | 01 | 0 | 1 | 1 | 0 |
| | 11 | 1 | 1 | 1 | 0 |
| | 10 | 1 | 0 | 0 | 1 |

| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 1 | 0 | 1 | 0 |
| | 01 | 1 | 0 | 0 | 1 |
| | 11 | 0 | 0 | 0 | 1 |
| | 10 | 0 | 1 | 1 | 0 |

Se muestran los implicantes primos, y el mapa de la función \bar{F} . El resultado es: $\bar{F} = \bar{A} \cdot \bar{B} \cdot \bar{C} + B \cdot C \cdot \bar{D} + A \cdot B \cdot \bar{D} + A \cdot \bar{B} \cdot D$

$$\begin{aligned}
 F &= \bar{\bar{F}} = \overline{\bar{A} \cdot \bar{B} \cdot \bar{C} + B \cdot C \cdot \bar{D} + A \cdot B \cdot \bar{D} + A \cdot \bar{B} \cdot D} = \\
 &= (A + B + C) \cdot (\bar{B} + \bar{C} + D) \cdot (\bar{A} + \bar{B} + D) \cdot (\bar{A} + B + \bar{D})
 \end{aligned}$$

Ejemplo de Simplificación por ceros.

$$F = (A + B + C) \cdot (\bar{B} + \bar{C} + D) \cdot (\bar{A} + \bar{B} + D) \cdot (\bar{A} + B + \bar{D})$$

| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 0 | 1 | 0 | 1 |
| | 01 | 0 | 1 | 1 | 0 |
| | 11 | 1 | 1 | 1 | 0 |
| | 10 | 1 | 0 | 0 | 1 |

Observaciones:

- La variable que se elimina sigue siendo la que cambia
- De las variables que no cambian, aparecen negadas cuando en las casillas correspondientes aparezca a 1 y sin negar cuando aparezca a 0
- La expresión obtenida es de producto de sumas

Salidas no determinadas

$F(A,B,C,D)$, que valga uno cuando el dígito decimal a la entrada (4 bits), interpretada en binario natural, esté entre 4 y 8 (ambos inclusive):

| ENTRADAS | | | | SALIDA |
|----------|---|---|---|--------|
| A | B | C | D | F |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | X |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | 0 | X |
| 1 | 1 | 0 | 1 | X |
| 1 | 1 | 1 | 0 | X |
| 1 | 1 | 1 | 1 | X |

| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 0 | 1 | X | 1 |
| | 01 | 0 | 1 | X | 0 |
| | 11 | 0 | 1 | X | X |
| | 10 | 0 | 1 | X | X |

$$F(A, B, C, D) = B + A \cdot \overline{D}$$

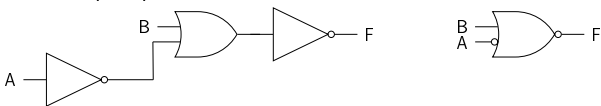
Agenda

- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**
 - Puertas lógicas básicas
 - Retardos
 - Implementación con puertas NAND/NOR
 - Bloques lógicos
 - Codificadores y decodificadores
 - Multiplexores
 - Memorias ROM y arrays lógicos programables
 - Unidad aritmético lógica

Circuitos Combinacionales Comunes

- **Circuito Combinacional (sin memoria):** Las salidas dependen sólo de las entradas actuales. Ejemplo: ALU
- **Circuito Secuencial (con memoria):** Las salidas dependen de las entradas y del valor almacenado en su memoria (estado). Ejemplo: Memoria de Datos.
- Implementación con puertas lógicas de $F = \overline{\overline{A} + B}$: Utilizando explícitamente inversores (izq) y utilizando entradas y salidas con burbujas (der):



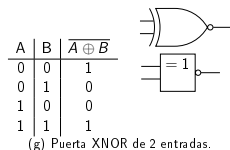
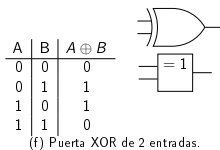
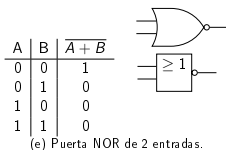
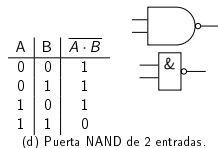
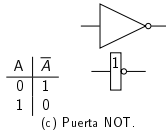
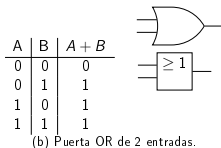
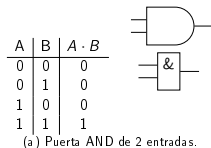
Agenda

- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**

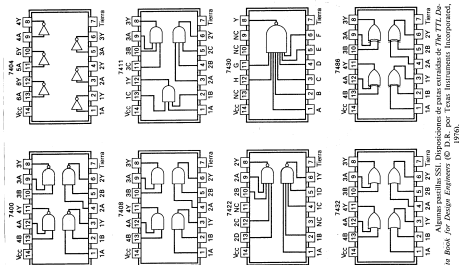
- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- Bloques lógicos
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Puertas lógicas básicas



Circuitos integrados

- SSI (circuitos integrados a escala pequeña): 1 a 10 puertas
- MSI (circuitos integrados a escala media): 10 a 100 puertas
- LSI (circuitos integrados a escala grande): 100 a 100.000 puertas
- VLSI (circuitos integrados a escala muy grande): Más de 100.000 puertas
- Ejemplos de CI SSI:



Agenda

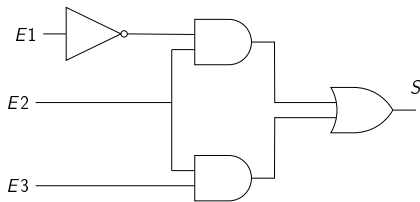
- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**

- Puertas lógicas básicas
- **Retardos**
- Implementación con puertas NAND/NOR
- Bloques lógicos
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Retardos

- Tiempo que transcurre entre el instante en que un circuito tiene disponibles los valores de señal deseados a la entrada y el instante en que la señal de salida se estabiliza al valor deseado
- Ejemplo: NOT 5 ns, AND 10 ns, OR 10 ns:
 - Tiempo total del circuito de ejemplo = $5+10+10=25\text{ns}$
 - El AND inferior trabaja en paralelo con la parte superior del circuito.



Agenda

- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**
 - Puertas lógicas básicas
 - Retardos
 - **Implementación con puertas NAND/NOR**
 - Bloques lógicos
 - Codificadores y decodificadores
 - Multiplexores
 - Memorias ROM y arrays lógicos programables
 - Unidad aritmético lógica

Implementación con puertas NAND/NOR

- Para implementar con puertas NAND, simplificamos por unos. Sea $F = \sum m(2, 3, 4, 5, 7, 8, 10, 13, 15)$, entonces

$$F(A, B, C, D) = \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + B \cdot D + A \cdot \bar{B} \cdot \bar{D}$$

si a continuación negamos dos veces y aplicamos De Morgan obtenemos:

$$\begin{aligned} F(A, B, C, D) &= \overline{\overline{\bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + B \cdot D + A \cdot \bar{B} \cdot \bar{D}}} \\ &= \overline{(\bar{A} \cdot \bar{B} \cdot C) \cdot (\bar{A} \cdot B \cdot \bar{C}) \cdot (B \cdot D) \cdot (A \cdot \bar{B} \cdot \bar{D})} \end{aligned}$$

que puede implementarse directamente mediante puertas NAND.

Implementación con puertas NAND/NOR

- Para implementar con puertas NOR, simplificamos por ceros.
Sea $F = \sum m(2, 3, 4, 5, 7, 8, 10, 13, 15)$, entonces

$$F(A, B, C, D) = (A+B+C) \cdot (\bar{B} + \bar{C} + D) \cdot (\bar{A} + \bar{B} + D) \cdot (\bar{A} + B + \bar{D})$$

si a continuación negamos dos veces obtenemos y aplicamos De Morgan obtenemos:

$$\begin{aligned} & \overline{\overline{(A+B+C) \cdot (\bar{B} + \bar{C} + D) \cdot (\bar{A} + \bar{B} + D) \cdot (\bar{A} + B + \bar{D})}} \\ & = \overline{\overline{(A+B+C)} + \overline{\overline{(\bar{B} + \bar{C} + D)}} + \overline{\overline{(\bar{A} + \bar{B} + D)}} + \overline{\overline{(\bar{A} + B + \bar{D})}}} \end{aligned}$$

que puede implementarse directamente mediante puertas NOR.

Agenda

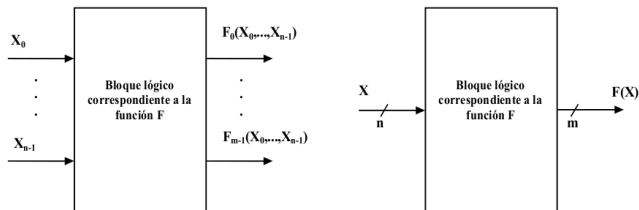
- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**

- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- **Bloques lógicos**
- Codificadores y decodificadores
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

Bloques lógicos

- Conforme construimos funciones lógicas cada vez más complejas se hace inviable representar gráficamente el diagrama de conexión completo con todas las puertas resultantes.
- Utilizaremos bloques lógicos para encapsular y ocultar la complejidad de igual manera que utilizamos funciones y procedimientos al programar:



Agenda

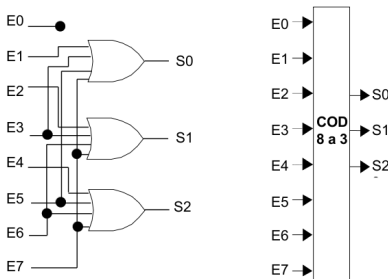
- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**

- Puertas lógicas básicas
- Retardos
- Implementación con puertas NAND/NOR
- Bloques lógicos
- **Codificadores y decodificadores**
- Multiplexores
- Memorias ROM y arrays lógicos programables
- Unidad aritmético lógica

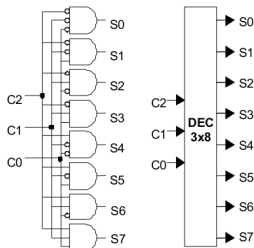
Codificadores y decodificadores

- **Codificador:** Circuito con 2^n líneas de entrada y n líneas de salida.
- Una y sólo una línea de entrada se activa en cada momento. En la salida aparece, codificado en binario, el número de salida activada.
- Es sencillo generalizar para cualquier número de entradas.



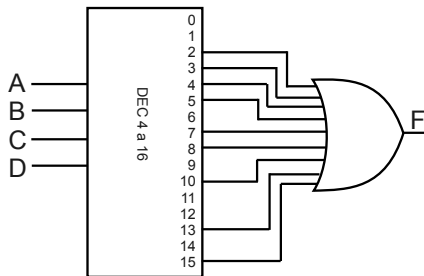
Codificadores y decodificadores

- **Decodificador:** Circuito con n líneas de entrada y 2^n líneas de salida.
- Una y sólo una línea de salida se activa en cada momento. La salida activada es la correspondiente al número binario codificado en la entrada (es un generador de mintérminos).
- También puede generalizarse para cualquier número de entradas.
- Empleado para direccionar posiciones de memoria.



Implementación de funciones con decodificadores

- Sea la función $F(A, B, C, D) = \sum m(2, 3, 4, 5, 7, 8, 10, 13, 15)$, utilizamos un decodificador de 4 a 16, conectando las 4 variables a las 4 entradas del decodificador y conectando a la puerta OR las salidas 2, 3, 4, 5, 7, 8, 10, 13 y 15 del decodificador:



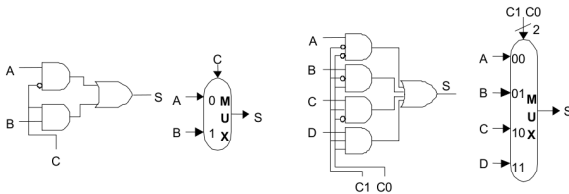
Agenda

- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**
 - Puertas lógicas básicas
 - Retardos
 - Implementación con puertas NAND/NOR
 - Bloques lógicos
 - Codificadores y decodificadores
 - **Multiplexores**
 - Memorias ROM y arrays lógicos programables
 - Unidad aritmético lógica

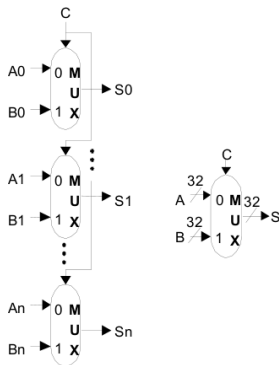
Multiplexores

- **Multiplexores:** 2^n líneas de entrada de datos, n líneas de entrada de control, una sola salida.
- Funciona como un selector de datos: Las n líneas de control seleccionan aquella entrada de datos que se deja pasar hasta la salida.
- También puede generalizarse para cualquier número de entradas. Ejemplos: MUX 2x1 y MUX 4x1.



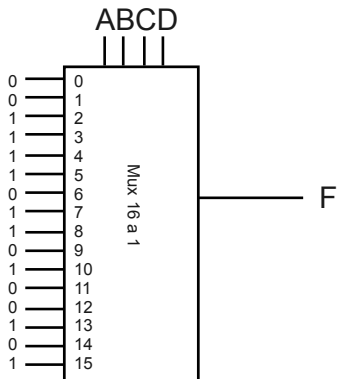
Multiplexores

- Otra posible extensión de los multiplexores es en el ancho de la palabra seleccionada (ancho de entradas de datos y de la salida).
- Ejemplo: MUX 2x1 de 32 bits de ancho, usando 32 MUX 2x1 de 1 bit:



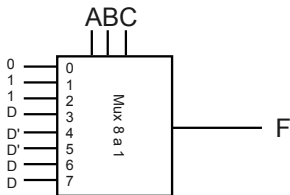
Implementación de Funciones con Multiplexores

- Sea la función $F(A, B, C, D) = \sum m(2, 3, 4, 5, 7, 8, 10, 13, 15)$, utilizamos un multiplexor 16 a 1, conectando las 4 variables a las 4 entradas de control y poniendo en las entradas los 16 valores de la tabla de verdad:



Implementación de Funciones con Multiplexores

- Si utilizamos un multiplexor de 8 a 1, a cada entrada le corresponde dos entradas de la tabla de verdad con cuatro posibles valores: (0,0), (0,1), (1,0) y (1,1).
- Conectaremos 3 de las variables a las entradas de control y pondremos en las entradas el valor 0, 1, la variable excluida o su negación dependiendo de esas cuatro posibilidades:



| | | AB | | | |
|----|----|----|----|----|----|
| | | 00 | 01 | 11 | 10 |
| CD | 00 | 0 | 1 | 0 | 1 |
| | 01 | 0 | 1 | 1 | 0 |
| | 11 | 1 | 1 | 1 | 0 |
| | 10 | 1 | 0 | 0 | 1 |

Agenda

- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

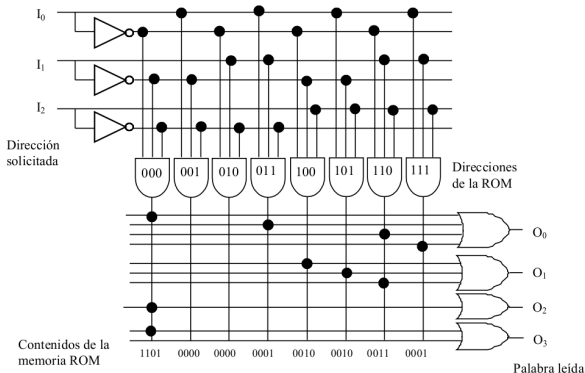
- 2 **Circuitos Combinacionales Comunes**
 - Puertas lógicas básicas
 - Retardos
 - Implementación con puertas NAND/NOR
 - Bloques lógicos
 - Codificadores y decodificadores
 - Multiplexores
 - **Memorias ROM y arrays lógicos programables**
 - Unidad aritmético lógica

Memorias ROM

- Memorias ROM (*Read Only Memory*, memoria de sólo lectura).
- m entradas (2^m elementos direccionables, o altura de la ROM).
- n salidas (Cada posición contiene un dato de n bits, anchura de la ROM).
- Forma de la ROM = altura x anchura
- Aunque se llame memoria, es un circuito combinacional.
- Puede usarse para implementar n funciones binarias distintas dependientes de las mismas m variables de entrada.
- Se implementa usando dos niveles de puertas (aparte de las negaciones de las entradas):
 - Un plano AND, con 2^m puertas de m entradas cada una.
 - Un plano OR, con n puertas de salida.

Memorias ROM

- Esquema de una memoria ROM con 8 posiciones de 4 bits cada una (3 bits de dirección, y anchura de datos 4).



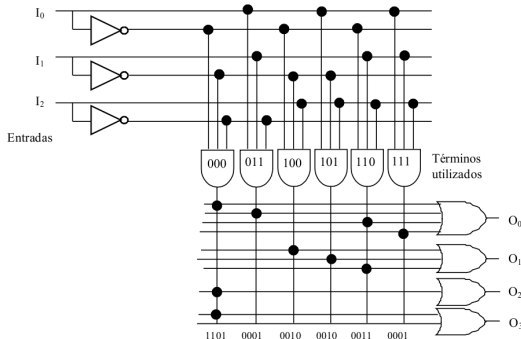
| I2 | I1 | I0 | O3 | O2 | O1 | O0 |
|----|----|----|----|----|----|----|
| 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |

Variantes de Memorias ROM

- PROM (*Programmable ROM*, ROM programables).
- EPROM (*Erasable PROM*, PROM borrables).
- EEPROM (*Electrically EPROM*, PROM borrables electrónicamente).
- Memorias Flash (permiten el borrado y reescritura selectivos por bloques, miles de veces).

Arrays Lógicos Programables

- PLA (*Programmable Logic Array*, array lógico programable).
- Como una ROM, pero sólo se implementan los productos necesarios. Útiles cuando hay muchas entradas, pero sólo unas pocas combinaciones se utilizan realmente. Ejemplo Forma 3x6x4:



Agenda

- 1 **Introducción**
 - Álgebra de Boole. Funciones lógicas
 - Mapas de Karnaugh

- 2 **Circuitos Combinacionales Comunes**
 - Puertas lógicas básicas
 - Retardos
 - Implementación con puertas NAND/NOR
 - Bloques lógicos
 - Codificadores y decodificadores
 - Multiplexores
 - Memorias ROM y arrays lógicos programables
 - **Unidad aritmético lógica**

Unidad aritmético lógica

- ALU (*Arithmetic-Logic Unit*, unidad aritmético lógica).
- Circuito capaz de hacer operaciones aritméticas (sumas, restas, etc.) o lógicas bit a bit (AND, OR, etc.) con dos palabras de entrada de una longitud dada (32 bits en MIPS).
- La operación concreta a realizar depende de los bits de operación.
- Pueden construirse ALUs enteras o de punto flotante.
- Estudiaremos sólo una sencilla ALU entera con suma, resta, comparación, AND y OR (no incluye multiplicación ni división).

