DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Sistemas Digitales

Circuitos Combinatorios

Primer Cuatrimestre 2024

Sistemas Digitales
DC - UBA

Repaso: Algebra de Boole

Axiomas

Partimos de las siguientes proposiciones (axiomas):

Axiomas

Partimos de las siguientes proposiciones (axiomas):

(A1) Existen dos elementos: $X = 1$ *si* $X \neq 0$ *ó* $X = 0$ *si* $X \neq 1$

Axiomas

Partimos de las siguientes proposiciones (axiomas):

(A1) Existen dos elementos: $X = 1$ *si* $X \neq 0$ *ó* $X = 0$ *si* $X \neq 1$

(A2) Existe el operador negación $\overline{()}$ tal que: *Si* $X = 1 \Rightarrow \overline{X} = 0$

Axiomas

Partimos de las siguientes proposiciones (axiomas):

(A1) Existen dos elementos: $X = 1$ *si* $X \neq 0$ *ó* $X = 0$ *si* $X \neq 1$

(A2) Existe el operador negación $\overline{()}$ tal que: *Si* $X = 1 \Rightarrow \overline{X} = 0$

(A3) $0 \cdot 0 = 0 \qquad 1 + 1 = 1$

Axiomas

Partimos de las siguientes proposiciones (axiomas):

(A1) Existen dos elementos: $X = 1$ *si* $X \neq 0$ *ó* $X = 0$ *si* $X \neq 1$

(A2) Existe el operador negación $\overline{()}$ tal que: *Si* $X = 1 \Rightarrow \overline{X} = 0$

(A3) $0 \cdot 0 = 0 \qquad 1 + 1 = 1$

(A4) $1 \cdot 1 = 1 \qquad 0 + 0 = 0$

Axiomas

Partimos de las siguientes proposiciones (axiomas):

(A1) Existen dos elementos: $X = 1$ *si* $X \neq 0$ *ó* $X = 0$ *si* $X \neq 1$

(A2) Existe el operador negación $\overline{()}$ tal que: *Si* $X = 1 \Rightarrow \overline{X} = 0$

(A3) $0 \cdot 0 = 0 \qquad 1 + 1 = 1$

(A4) $1 \cdot 1 = 1 \qquad 0 + 0 = 0$

(A5) $0 \cdot 1 = 1 \cdot 0 = 0 \qquad 0 + 1 = 1 + 0 = 1$

Propiedades

De los axiomas anteriores se derivan las siguientes propiedades:

Propiedad	AND	OR
Identidad	$1.A = A$	$0 + A = A$
Nulo	$0.A = 0$	$1 + A = 1$
Idempotencia	$A.A = A$	$A + A = A$
Inverso	$A.\overline{A} = 0$	$A + \overline{A} = 1$
Conmutatividad	$A.B = B.A$	$A + B = B + A$
Asociatividad	$(A.B).C = A.(B.C)$	$(A + B) + C = A + (B + C)$
Distributividad	$A + (B.C) = (A + B).(A + C)$	$A.(B + C) = A.B + A.C$
Absorción	$A.(A + B) = A$	$A + A.B = A$
De Morgan	$\overline{A.B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$

Propiedades

De los axiomas anteriores se derivan las siguientes propiedades:

Propiedad	AND	OR
Identidad	$1.A = A$	$0 + A = A$
Nulo	$0.A = 0$	$1 + A = 1$
Idempotencia	$A.A = A$	$A + A = A$
Inverso	$A.\overline{A} = 0$	$A + \overline{A} = 1$
Conmutatividad	$A.B = B.A$	$A + B = B + A$
Asociatividad	$(A.B).C = A.(B.C)$	$(A + B) + C = A + (B + C)$
Distributividad	$A + (B.C) = (A + B).(A + C)$	$A.(B + C) = A.B + A.C$
Absorción	$A.(A + B) = A$	$A + A.B = A$
De Morgan	$\overline{A.B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$

Tarea: ¡Demostrarlas!

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Solución:

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)} \longleftarrow$ De Morgan

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Solución:

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)} \longleftarrow$ De Morgan

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z} \longleftarrow$ Distributiva

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Solución:

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)} \longleftarrow$ De Morgan

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z} \longleftarrow$ Distributiva

$\overline{(\overline{X} \cdot Y)} \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ De Morgan

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Solución:

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)} \longleftarrow$ De Morgan

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z} \longleftarrow$ Distributiva

$(\overline{X} \cdot Y) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ De Morgan

$(X + \overline{Y}) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ Distributiva

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Solución:

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)} \longleftarrow$ De Morgan

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z} \longleftarrow$ Distributiva

$(\overline{X} \cdot Y) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ De Morgan

$(X + \overline{Y}) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ Distributiva

$(X + \overline{Y}) \cdot (Z + \overline{Z}) \longleftarrow$ Inverso

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}$$

Solución:

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)} \longleftarrow$ De Morgan

$\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z} \longleftarrow$ Distributiva

$(\overline{X} \cdot Y) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ De Morgan

$(X + \overline{Y}) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ Distributiva

$(X + \overline{Y}) \cdot (Z + \overline{Z}) \longleftarrow$ Inverso

$(X + \overline{Y}) \cdot 1 \longleftarrow$ Identidad

Ejercicio I

Demostrar si la siguiente igualdad entre funciones booleanas es verdadera o falsa:

$$(X + \overline{Y}) = \overline{\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}}$$

Solución:

$\overline{\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{(Y + Z)}} \longleftarrow$ De Morgan

$\overline{\overline{(\overline{X} \cdot Y)} \cdot Z + X \cdot \overline{Z} + \overline{Y} \cdot \overline{Z}} \longleftarrow$ Distributiva

$\overline{(\overline{X} \cdot Y)} \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ De Morgan

$(X + \overline{Y}) \cdot Z + (X + \overline{Y}) \cdot \overline{Z} \longleftarrow$ Distributiva

$(X + \overline{Y}) \cdot (Z + \overline{Z}) \longleftarrow$ Inverso

$(X + \overline{Y}) \cdot 1 \longleftarrow$ Identidad

$X + \overline{Y}$ *lqqd*.

Notación

En el lenguaje coloquial vamos a llamar a las operaciones indistintamente de la siguiente forma:

$$A + B \equiv A \ OR \ B$$

Notación

En el lenguaje coloquial vamos a llamar a las operaciones indistintamente de la siguiente forma:

$$A + B \equiv A \ OR \ B$$

$$AB \equiv A.B \equiv A \ AND \ B$$

Notación

En el lenguaje coloquial vamos a llamar a las operaciones indistintamente de la siguiente forma:

$$A + B \equiv A \ OR \ B$$

$$AB \equiv A.B \equiv A \ AND \ B$$

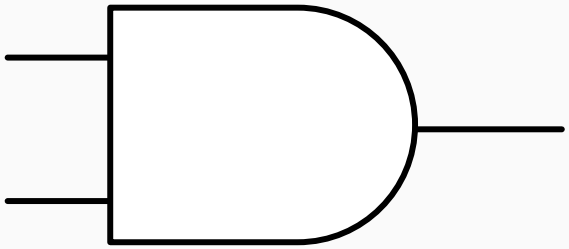
$$\overline{A} \equiv NOT \ A$$

Compuertas, señales y tablas de verdad

Compuertas

Son modelos idealizados de dispositivos electrónicos o de computo, que realizan operaciones booleanas.

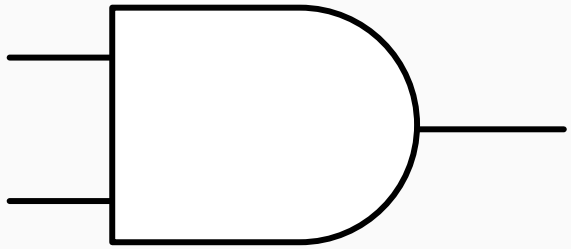
Las podemos representar gráficamente:



Compuertas

Son modelos idealizados de dispositivos electrónicos o de computo, que realizan operaciones booleanas.

Las podemos representar gráficamente:



O describir mediante lenguaje, por ejemplo en `SystemVerilog`:

```
assign o = a & b;
```

Son representaciones que nos permiten observar todas las salidas para todas las combinaciones de entradas[1].

Por ejemplo, la función del ejercicio ($F = X + \overline{Y}$) se representa:

X	Y	F
0	0	
0	1	
1	0	
1	1	

[1] Como resulta esperable, esta representación puede volverse muy compleja cuando el número de variables y salidas crece.

Son representaciones que nos permiten observar todas las salidas para todas las combinaciones de entradas[1].

Por ejemplo, la función del ejercicio ($F = X + \overline{Y}$) se representa:

X	Y	F
0	0	1
0	1	
1	0	
1	1	

[1] Como resulta esperable, esta representación puede volverse muy compleja cuando el número de variables y salidas crece.

Son representaciones que nos permiten observar todas las salidas para todas las combinaciones de entradas[1].

Por ejemplo, la función del ejercicio ($F = X + \overline{Y}$) se representa:

X	Y	F
0	0	1
0	1	0
1	0	
1	1	

[1] Como resulta esperable, esta representación puede volverse muy compleja cuando el número de variables y salidas crece.

Son representaciones que nos permiten observar todas las salidas para todas las combinaciones de entradas[1].

Por ejemplo, la función del ejercicio ($F = X + \overline{Y}$) se representa:

X	Y	F
0	0	1
0	1	0
1	0	1
1	1	

[1] Como resulta esperable, esta representación puede volverse muy compleja cuando el número de variables y salidas crece.

Tablas de verdad

Son representaciones que nos permiten observar todas las salidas para todas las combinaciones de entradas[1].

Por ejemplo, la función del ejercicio ($F = X + \overline{Y}$) se representa:

X	Y	F
0	0	1
0	1	0
1	0	1
1	1	1

[1] Como resulta esperable, esta representación puede volverse muy compleja cuando el número de variables y salidas crece.

Compuertas - NOT

Gráficamente:

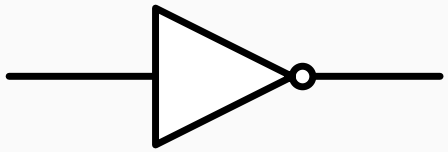


Tabla de verdad:

A	NOT A
0	1
1	0

En `SystemVerilog`:

```
assign o = ~a;
```

Compuertas - AND

Gráficamente:

Tabla de verdad:

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

En `SystemVerilog`:

```
assign o = a & b;
```

Compuertas - OR

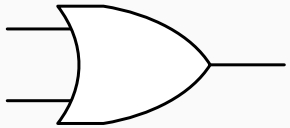
Gráficamente: 

Tabla de verdad:

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

En `SystemVerilog`:

```
assign o = a | b;
```

Compuertas - XOR u OR-EXCLUSIVA

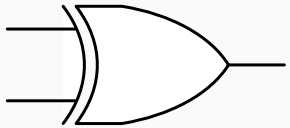
Gráficamente: 

Tabla de verdad:

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	0	0

En `SystemVerilog`:

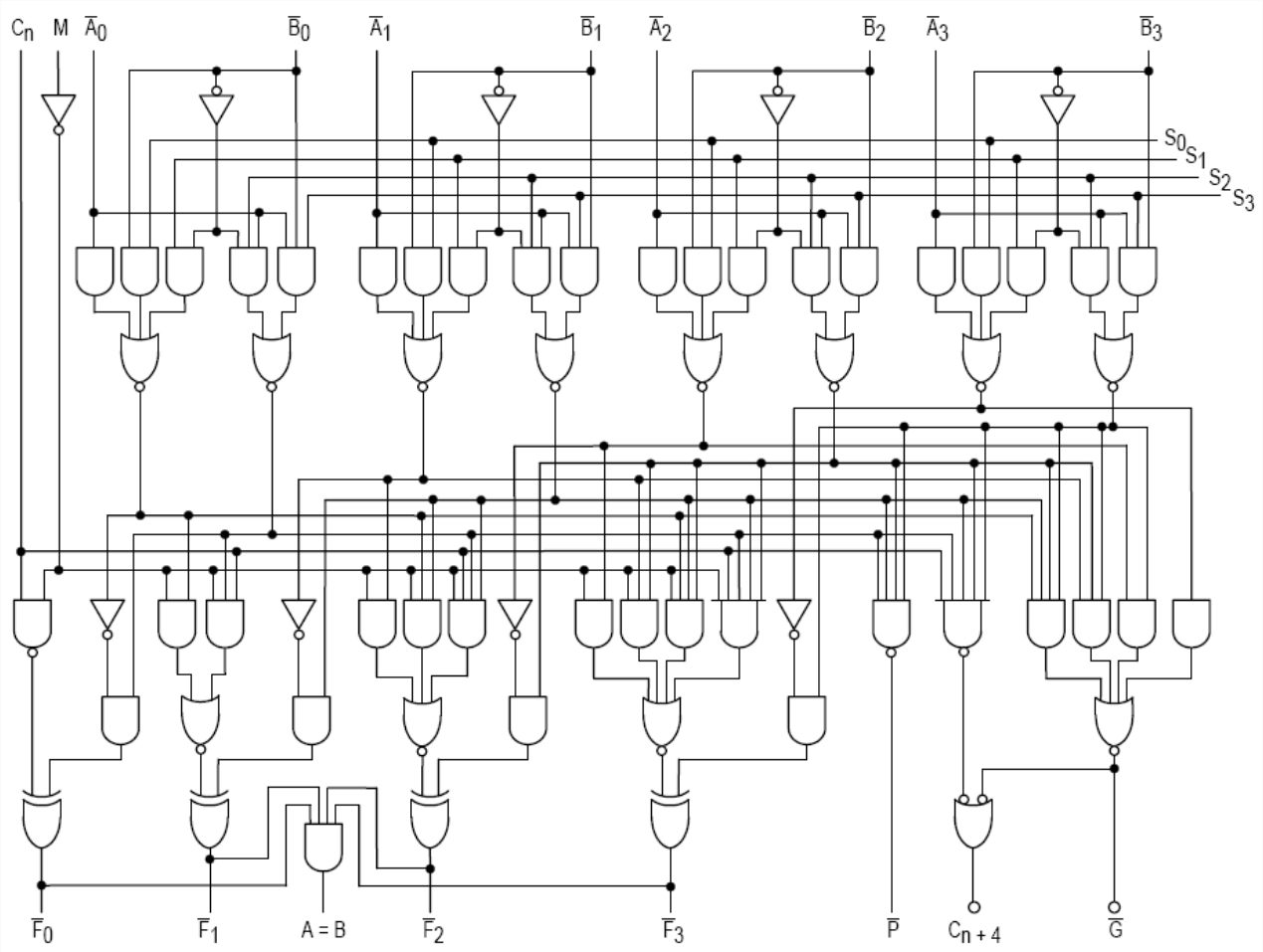
```
assign o = a ^ b;
```

Entradas y salidas - Categorización

Distintas vistas de un circuito

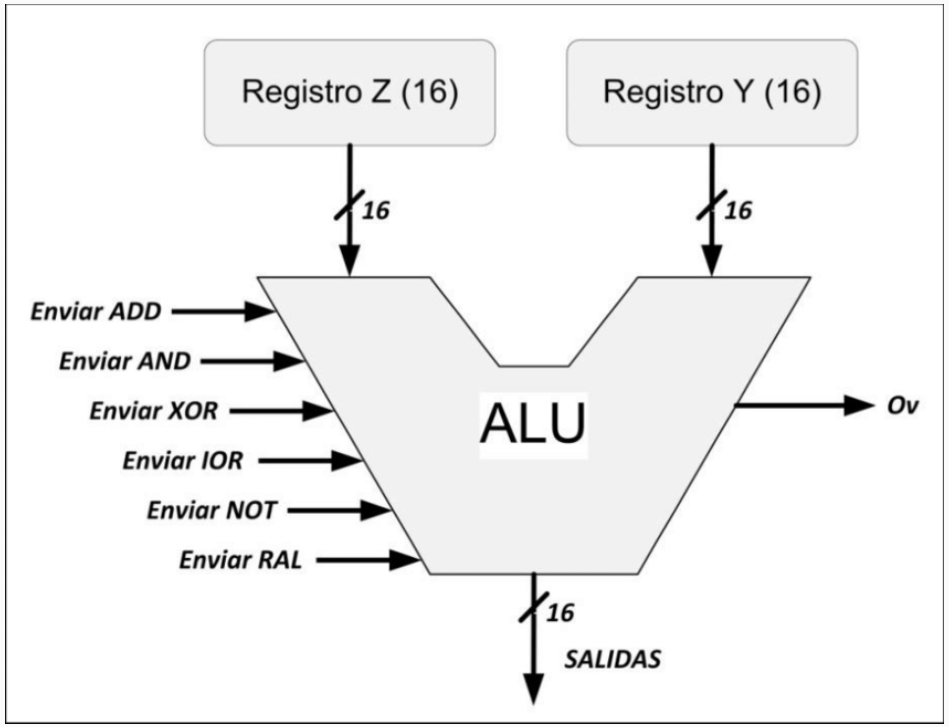
Por momentos vamos a querer abstraer nuestros circuitos en módulos de los cuáles observaremos solamente sus entradas y salidas. Veamos un ejemplo donde ocultamos parte de la complejidad pasando de una vista interna del circuito (caja blanca) a una externa (caja negra).

Desde el museo: ALU 74181



ALU revisitada

Aplicando lo anterior, podemos trabajar con la ALU viéndola de la siguiente manera:



Establecen el sentido de la información:

En la ALU anterior se representan con las flechas...

Establecen el sentido de la información:

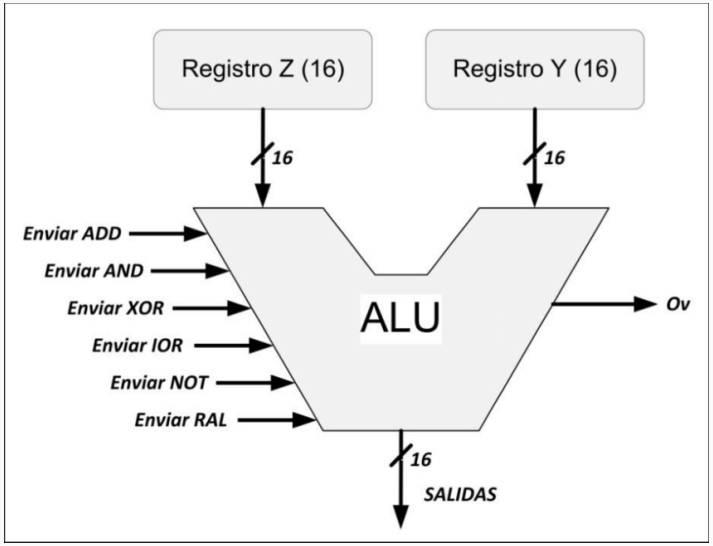
En la ALU anterior se representan con las flechas...

En `SystemVerilog`:

```
module ALU #(parameter DATA_WIDTH = 16)
            (input [DATA_WIDTH-1:0] operandoZ,
            input [DATA_WIDTH-1:0] operandoY,
            input [2:0] opcode,
            output [DATA_WIDTH-1:0] salidas,
            output overflow);
end module;
```

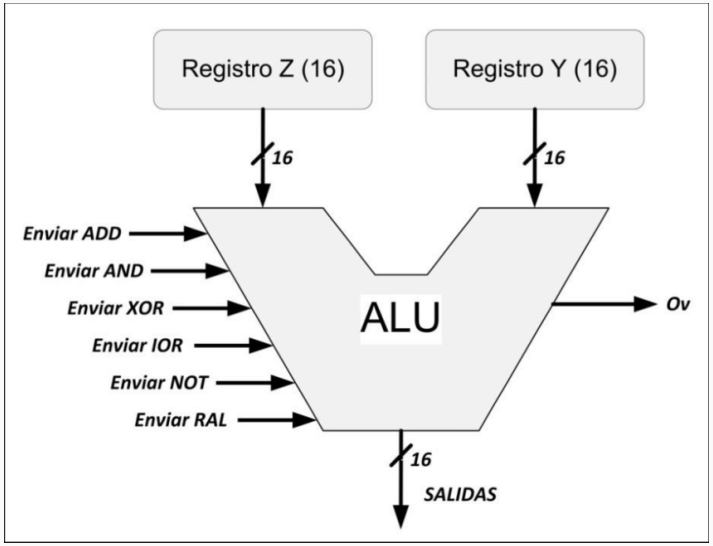
Entradas/Salidas: Tipos

En la ALU, ¿son funcionalmente todas iguales las entradas y salidas?



Entradas/Salidas: Tipos

En la ALU, ¿son funcionalmente todas iguales las entradas y salidas?



NO

Datos vs. Control

Lógica proposicional a circuitos combinatorios

Motivación

El estudio de la lógica proposicional y del álgebra de Boole tiene que ver con que vamos a querer implementar funciones lógicas en nuestro soporte electrónico con circuitos combinatorios.

Traduciendo una fórmula

Sabemos que se puede describir el comportamiento de un circuito combinatorio construyendo una tabla de verdad que determine las salidas que corresponden a cada combinación de los valores de entrada. Vamos a utilizar esto para describir un procedimiento que nos permite construir un circuito combinatorio cuyo comportamiento implementa cualquier fórmula proposicional φ.

Traduciendo una fórmula

Habrá casos en los que nos resultará difícil derivar un circuito de la fórmula, ya sea porque no vemos un vínculo directo entre la expresión y las compuertas básicas, o porque es conveniente expresarlo con una tabla de verdad.

Mecanismo de traducción

El mecanismo es el siguiente:

Mecanismo de traducción

El mecanismo es el siguiente:

- Si tenemos una fórmula φ que se expresa en función de las variables $x_1, \ldots, x_n$(las entradas).

Mecanismo de traducción

El mecanismo es el siguiente:

- Si tenemos una fórmula φ que se expresa en función de las variables $x_1, \ldots, x_n$(las entradas).
- Construimos una tabla de verdad con una fila para cada combinación posible de las entradas (por ej. $x_1 \to 1, x_2 \to 0, \ldots, x_n \to 1$) y en la columna de la salida y ingresamos el valor de la fórmula evaluada en esos valores $\varphi(1, 0, \ldots, 1)$.

Mecanismo de traducción

El mecanismo es el siguiente:

Mecanismo de traducción

El mecanismo es el siguiente:

- Vamos a utilizar solamente las filas en las que la función vale 1.

Mecanismo de traducción

El mecanismo es el siguiente:

- Vamos a utilizar solamente las filas en las que la función vale 1.
- Para cada fila i en la que φ es verdadera (vale 1) vamos a construir un término t_i como conjunción (y lógico) de todas las entradas, donde cada variable aparece negada si su valor era 0 en la fila y sin negar en caso contrario.

Mecanismo de traducción

El mecanismo es el siguiente:

- Vamos a utilizar solamente las filas en las que la función vale 1.
- Para cada fila i en la que φ es verdadera (vale 1) vamos a construir un término t_i como conjunción (y lógico) de todas las entradas, donde cada variable aparece negada si su valor era 0 en la fila y sin negar en caso contrario.
- Por ejemplo, si en la fila 4 la asignación (valuación) de las variables era $x_1 \to 1, x_2 \to 0, \ldots, x_n \to 1$, t_4 va a ser $x_1 \wedge \neg x_1 \wedge \ldots x_n$.

Mecanismo de traducción

El mecanismo es el siguiente:

Mecanismo de traducción

El mecanismo es el siguiente:

- Una vez que tenemos los términos $t_i, t_j, \ldots$ para cada fial en la que la función vale 1, vamos a hacer una disyunción (o lógico) de todos los términos $\varphi' = t_i \vee t_j \vee \ldots$.

Mecanismo de traducción

El mecanismo es el siguiente:

- Una vez que tenemos los términos $t_i, t_j, \ldots$ para cada fial en la que la función vale 1, vamos a hacer una disyunción (o lógico) de todos los términos $\varphi' = t_i \vee t_j \vee \ldots$.
- A esto se conoce como suma de productos y nos da una expresión de φ o de la tabla de verdad que puede traducirse fácilmente a un circuito combinatorio.

Volviendo al ejemplo

La fórmula $(F = X + \overline{Y})$ se representa:

X	Y	F
0	0	1
0	1	0
1	0	1
1	1	1

Volviendo al ejemplo

La fórmula $(F = X + \overline{Y})$ se representa:

X	Y	F
0	0	1
0	1	0
1	0	1
1	1	1

En este caso los términos serían $t_1 = \neg x \wedge \neg y, t_3 = \neg x \wedge y$ y $t_4 = x \wedge y$ y $\varphi' = (\neg x \wedge \neg y) \vee (\neg x \wedge y) \vee (x \wedge y)$.

A esta expresión se conoce como suma de productos.

Circuitos básicos

Ejercicio I - Inversor de 3 bits

Armar un circuito que invierta o no tres entradas de acuerdo al valor de una entrada adicional que actúa como control. En otras palabras, un inversor de k-bits es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = \overline{e_i}$ $\quad \forall i < k$
- Si $e_k = 0$, entonces $s_i = e_i$ $\quad \forall i < k$

Ejemplo:

inversor(1,011)=100

Ejercicio I - Inversor de 3 bits

Armar un circuito que invierta o no tres entradas de acuerdo al valor de una entrada adicional que actúa como control. En otras palabras, un inversor de k-bits es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = \overline{e_i}$ $\quad \forall i < k$
- Si $e_k = 0$, entonces $s_i = e_i$ $\quad \forall i < k$

Ejemplo:

inversor(1,011)=100 inversor(0,011)=011

Ejercicio I - Inversor de 3 bits

Armar un circuito que invierta o no tres entradas de acuerdo al valor de una entrada adicional que actúa como control. En otras palabras, un inversor de k-bits es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = \overline{e_i}$ $\quad \forall i < k$
- Si $e_k = 0$, entonces $s_i = e_i$ $\quad \forall i < k$

Ejemplo:

inversor(1,011)=100 inversor(0,011)=011
inversor(1,100)=011

Ejercicio I - Inversor de 3 bits

Armar un circuito que invierta o no tres entradas de acuerdo al valor de una entrada adicional que actúa como control. En otras palabras, un inversor de k-bits es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = \overline{e_i}$ $\quad \forall i < k$
- Si $e_k = 0$, entonces $s_i = e_i$ $\quad \forall i < k$

Ejemplo:

inversor(1,011)=100 inversor(0,011)=011
inversor(1,100)=011 inversor(1,101)=010

Ejercicio I - Inversor de 3 bits

¡Divide y conquista! Primero, con un bit...

ei	*ek*	*si*
0	0	0
0	1	1
1	0	1
1	1	0

Ejercicio I - Inversor de 3 bits

¡Divide y conquista! Primero, con un bit...

ei	ek	si
0	0	0
0	1	1
1	0	1
1	1	0

Como suma de productos,

$(\overline{ei} \cdot ek) + (ei \cdot \overline{ek})$

Ejercicio I - Inversor de 3 bits

¡Divide y conquista! Primero, con un bit...

ei	ek	si
0	0	0
0	1	1
1	0	1
1	1	0

Como suma de productos,

$(\overline{ei} \cdot ek) + (ei \cdot \overline{ek})$

¡Oh! casualidad, es una XOR ($\oplus$)

$(\overline{A} \cdot B) + (A \cdot \overline{B}) = A \oplus B$

Ejercicio I - Inversor de 3 bits

¡Divide y conquista! Primero, con un bit...

ei	ek	si
0	0	0
0	1	1
1	0	1
1	1	0

Como suma de productos,

$(\overline{ei} \cdot ek) + (ei \cdot \overline{ek})$

¡Oh! casualidad, es una XOR ($\oplus$)

$(\overline{A} \cdot B) + (A \cdot \overline{B}) = A \oplus B$

Extendiendo a 3 bits..

Ejercicio I - Inversor de 3 bits

¡Divide y conquista! Primero, con un bit...

ei	ek	si
0	0	0
0	1	1
1	0	1
1	1	0

Como suma de productos,

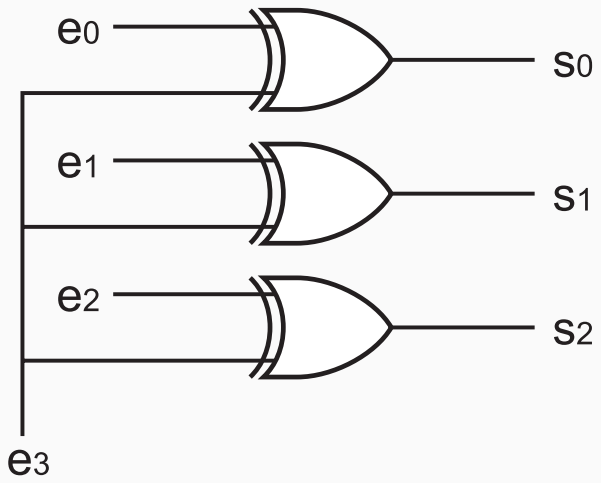
$(\overline{ei} \cdot ek) + (ei \cdot \overline{ek})$

¡Oh! casualidad, es una XOR ($\oplus$)

$(\overline{A} \cdot B) + (A \cdot \overline{B}) = A \oplus B$

Extendiendo a 3 bits..

Solucion con XOR:



Ejercicio II - Sumador Simple

Armar un **sumador de 1 bit**. Tiene que tener dos entradas de un bit y dos salidas, una para el resultado y otra para indicar si hubo o no acarreo.

Ejercicio II - Sumador Simple

Armar un **sumador de 1 bit**. Tiene que tener dos entradas de un bit y dos salidas, una para el resultado y otra para indicar si hubo o no acarreo.

Solución:

Ejercicio II - Sumador Simple

Armar un **sumador de 1 bit**. Tiene que tener dos entradas de un bit y dos salidas, una para el resultado y otra para indicar si hubo o no acarreo.

Solución:

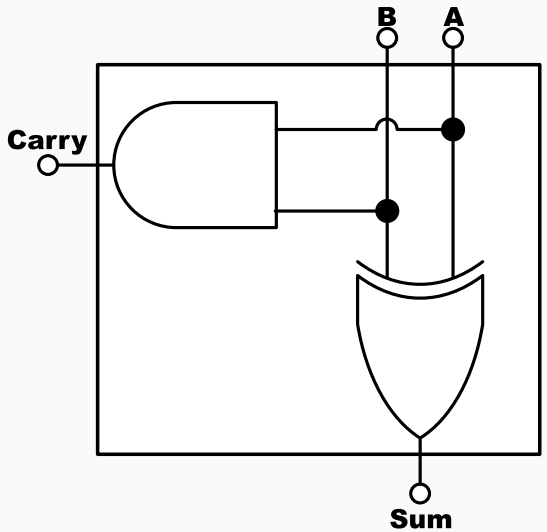
A	B	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Ejercicio II - Sumador Simple

Armar un **sumador de 1 bit**. Tiene que tener dos entradas de un bit y dos salidas, una para el resultado y otra para indicar si hubo o no acarreo.

Solución:

A	B	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Ejercicio III - Sumador Completo

Teniendo dos sumadores simples (de 1 bit) y sólo una compuerta a elección, arme un **sumador completo**. El mismo tiene 2 entradas de 1 bit y una tercer entrada interpretada como C_{In}, tiene como salida C_{Out} y S.

Ejercicio III - Sumador Completo

Teniendo dos sumadores simples (de 1 bit) y sólo una compuerta a elección, arme un **sumador completo**. El mismo tiene 2 entradas de 1 bit y una tercer entrada interpretada como C_{In}, tiene como salida C_{Out} y S. **Solución**:

Ejercicio III - Sumador Completo

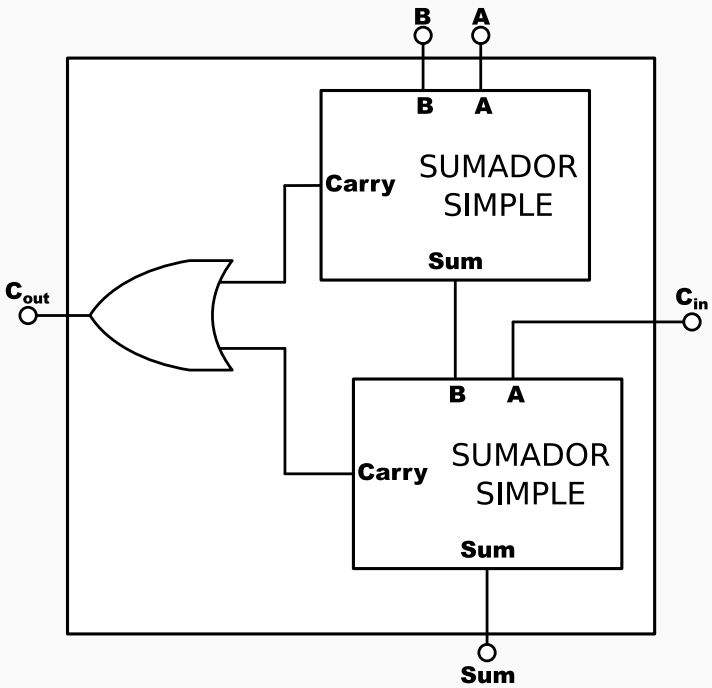
Teniendo dos sumadores simples (de 1 bit) y sólo una compuerta a elección, arme un **sumador completo**. El mismo tiene 2 entradas de 1 bit y una tercer entrada interpretada como C_{In}, tiene como salida C_{Out} y S. **Solución**:

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ejercicio III - Sumador Completo

Teniendo dos sumadores simples (de 1 bit) y sólo una compuerta a elección, arme un **sumador completo**. El mismo tiene 2 entradas de 1 bit y una tercer entrada interpretada como C_{In}, tiene como salida C_{Out} y S. **Solución**:

C_{in}	A	B	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Ejercicio IV - Sumador Completo de 3 bits

Armar un sumador completo de 3 bits.

Ejercicio IV - Sumador Completo de 3 bits

Armar un sumador completo de 3 bits.

Solución:

Ejercicio IV - Sumador Completo de 3 bits

Armar un sumador completo de 3 bits.

Solución:

¡Tarea!

Ejercicio IV - Shift

Armar un circuito de 3 *bits*. Este deberá mover a izquierda o a derecha los bits de entrada de acuerdo al valor de una entrada extra que actúa como control. En otras palabras, un shift *izq-der* de *k-bits* es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Ejercicio IV - Shift

Armar un circuito de 3 *bits*. Este deberá mover a izquierda o a derecha los bits de entrada de acuerdo al valor de una entrada extra que actúa como control. En otras palabras, un shift *izq-der* de *k-bits* es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k-1$ y $s_{k-1} = 0$

Ejemplos:

$$\text{shift\_lr}(1,011) = 110$$

Ejercicio IV - Shift

Armar un circuito de 3 *bits*. Este deberá mover a izquierda o a derecha los bits de entrada de acuerdo al valor de una entrada extra que actúa como control. En otras palabras, un shift *izq-der* de *k-bits* es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k-1$ y $s_{k-1} = 0$

Ejemplos:

$$\text{shift\_lr}(1,011) = 110 \quad \text{shift\_lr}(0,011) = 001$$

Ejercicio IV - Shift

Armar un circuito de 3 *bits*. Este deberá mover a izquierda o a derecha los bits de entrada de acuerdo al valor de una entrada extra que actúa como control. En otras palabras, un shift *izq-der* de *k-bits* es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Ejemplos:

$$\text{shift\_lr}(1,011) = 110 \quad \text{shift\_lr}(0,011) = 001$$
$$\text{shift\_lr}(1,100) = 000$$

Ejercicio IV - Shift

Armar un circuito de *3 bits*. Este deberá mover a izquierda o a derecha los bits de entrada de acuerdo al valor de una entrada extra que actúa como control. En otras palabras, un shift *izq-der* de *k-bits* es un circuito de $k+1$ entradas $(e_k, ..., e_0)$ y k salidas $(s_{k-1}, ..., s_0)$ que funciona del siguiente modo:

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Ejemplos:

shift_lr(1,011) = 110 shift_lr(0,011) = 001

shift_lr(1,100) = 000 shift_lr(1,101) = 010

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Solución:

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Solución:

$$s_2 = \begin{bmatrix} 0 & \text{si } e_3 = 0 \\ e_1 & \text{si } e_3 = 1 \end{bmatrix}$$

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Solución:

$$s_2 = \begin{bmatrix} 0 & \text{si } e_3 = 0 \\ e_1 & \text{si } e_3 = 1 \end{bmatrix}$$

$$e_3 . e_1$$

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Solución:

$$s_2 = \begin{bmatrix} 0 & \text{si } e_3 = 0 \\ e_1 & \text{si } e_3 = 1 \end{bmatrix} \qquad s_0 = \begin{bmatrix} 0 & \text{si } e_3 = 1 \\ e_1 & \text{si } e_3 = 0 \end{bmatrix}$$

$$e_3 . e_1$$

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Solución:

$$s_2 = \begin{bmatrix} 0 & \text{si } e_3 = 0 \\ e_1 & \text{si } e_3 = 1 \end{bmatrix} \qquad s_0 = \begin{bmatrix} 0 & \text{si } e_3 = 1 \\ e_1 & \text{si } e_3 = 0 \end{bmatrix}$$

$$e_3 . e_1 \qquad \overline{e_3} . e_1$$

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

Solución:

$$s_2 = \begin{bmatrix} 0 & \text{si } e_3 = 0 \\ e_1 & \text{si } e_3 = 1 \end{bmatrix} \quad s_0 = \begin{bmatrix} 0 & \text{si } e_3 = 1 \\ e_1 & \text{si } e_3 = 0 \end{bmatrix} \quad s_1 = \begin{bmatrix} e_0 & \text{si } e_3 = 1 \\ e_2 & \text{si } e_3 = 0 \end{bmatrix}$$

$$e_3 . e_1 \qquad \overline{e_3} . e_1$$

Ejercicio IV - Shift

- Si $e_k = 1$, entonces $s_i = e_{i-1}$ para todo $0 < i < k$ y $s_0 = 0$
- Si $e_k = 0$, entonces $s_i = e_{i+1}$ para todo $0 \leq i < k - 1$ y $s_{k-1} = 0$

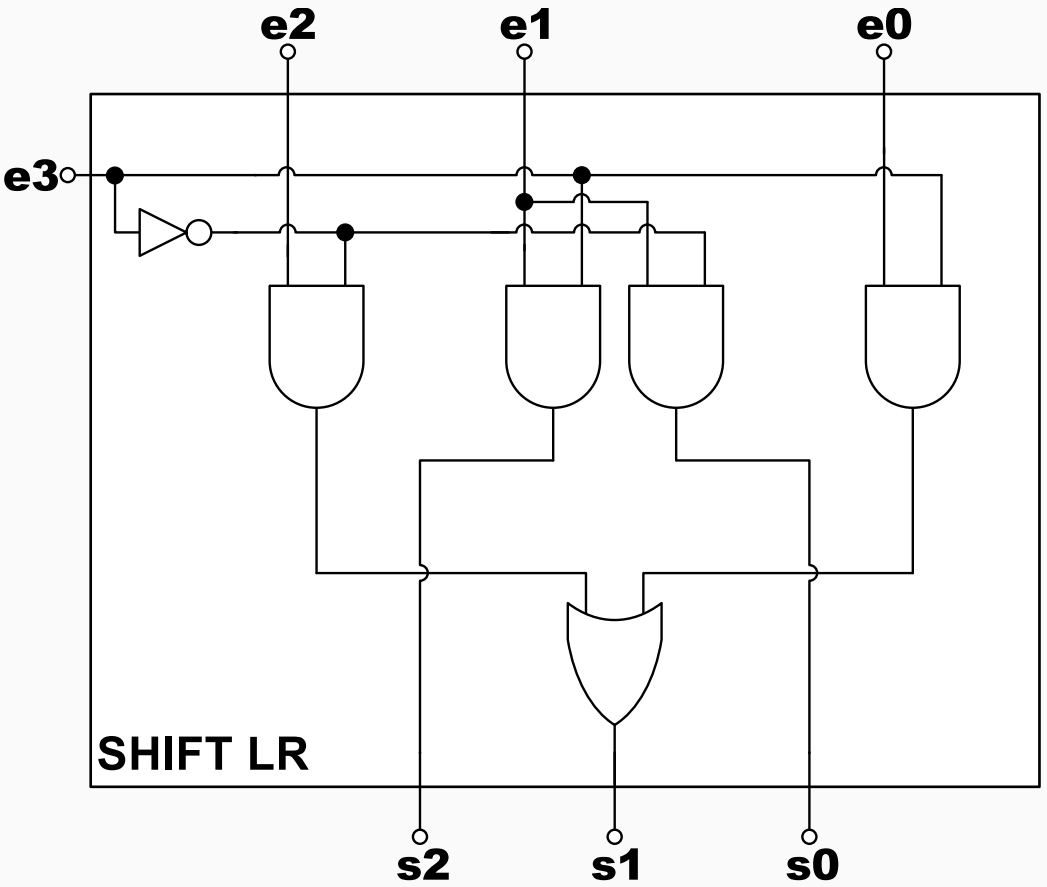
Solución:

$$s_2 = \begin{bmatrix} 0 & \text{si } e_3 = 0 \\ e_1 & \text{si } e_3 = 1 \end{bmatrix} \qquad s_0 = \begin{bmatrix} 0 & \text{si } e_3 = 1 \\ e_1 & \text{si } e_3 = 0 \end{bmatrix} \qquad s_1 = \begin{bmatrix} e_0 & \text{si } e_3 = 1 \\ e_2 & \text{si } e_3 = 0 \end{bmatrix}$$

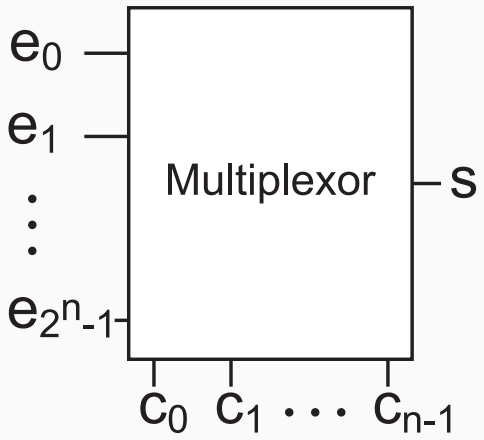
$e_3.e_1$ $\qquad$ $\overline{e_3}.e_1$ $\qquad$ $e_3.e_0 + \overline{e_3}.e_2$

Ejercicio IV - Shift

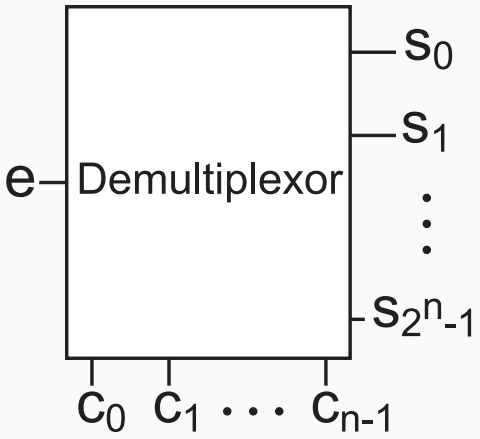
Solución:



Más combinatorios: Multiplexor y Demultiplexor



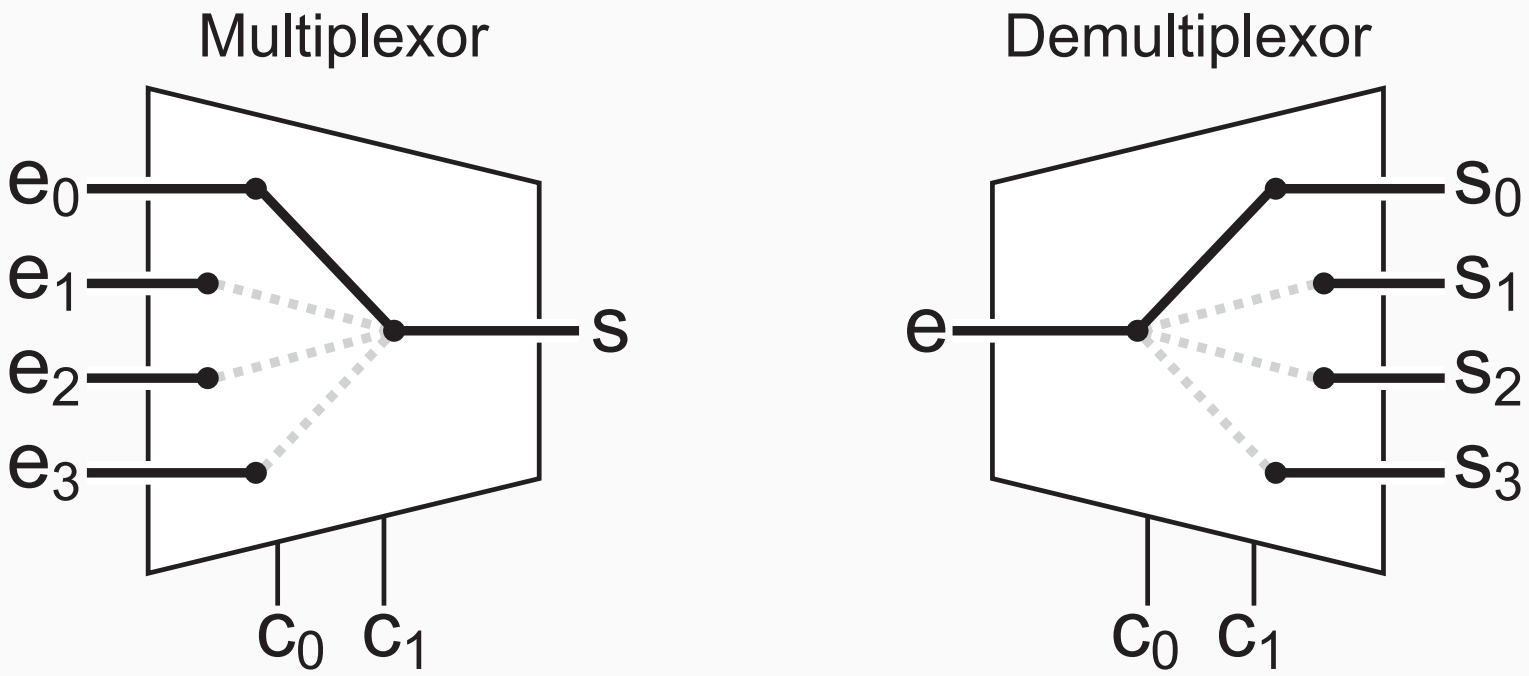
Las líneas de control c permiten seleccionar una de las entradas e, la que corresponderá a la salida s.



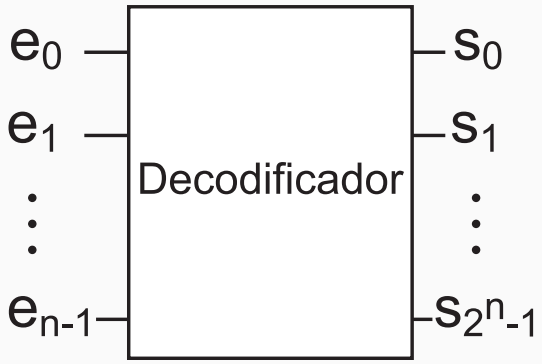
Las líneas de control c permiten seleccionar cual de las salidas s tendrá el valor de e.

Multiplexor y Demultiplexor

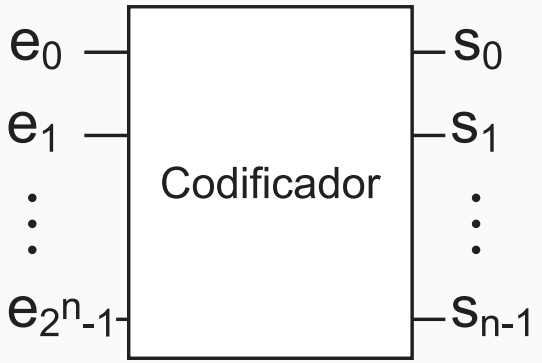
- Ejemplo,



Más combinatorios: Codificador y Decodificador



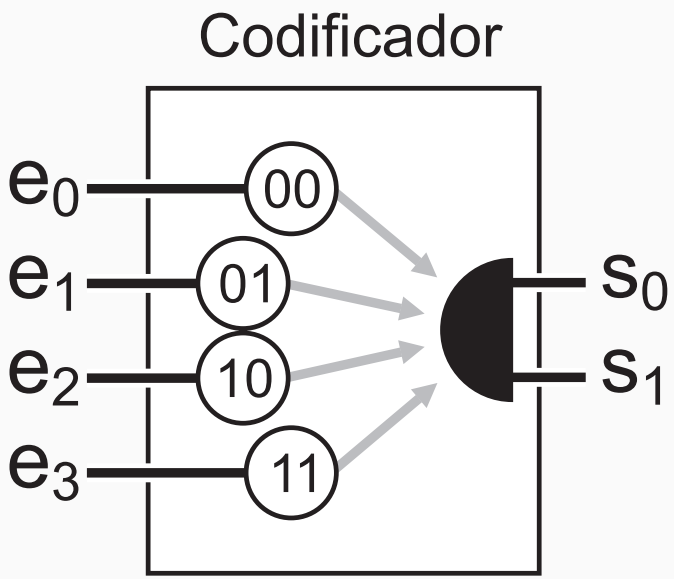
Cada combinación de las líneas e corresponderá a una sola línea en alto de la salida s.

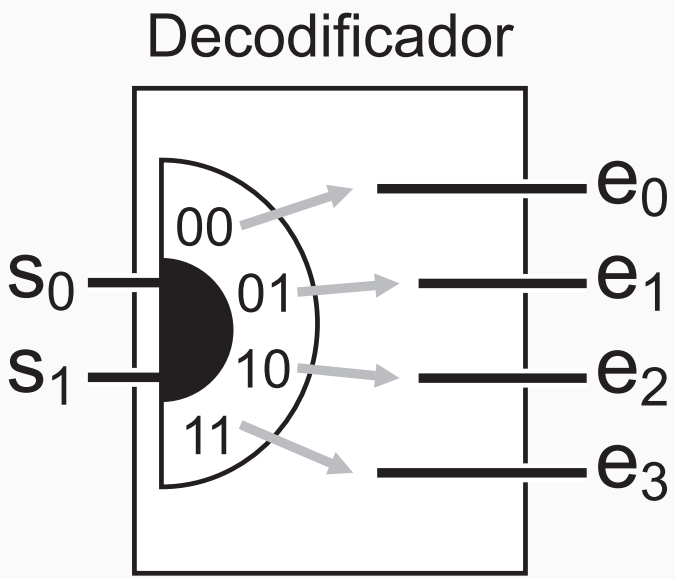


Una y sólo una línea en alto de e corresponderá a una combinación en la salida s.

Codificador y Decodificador

- Ejemplo,

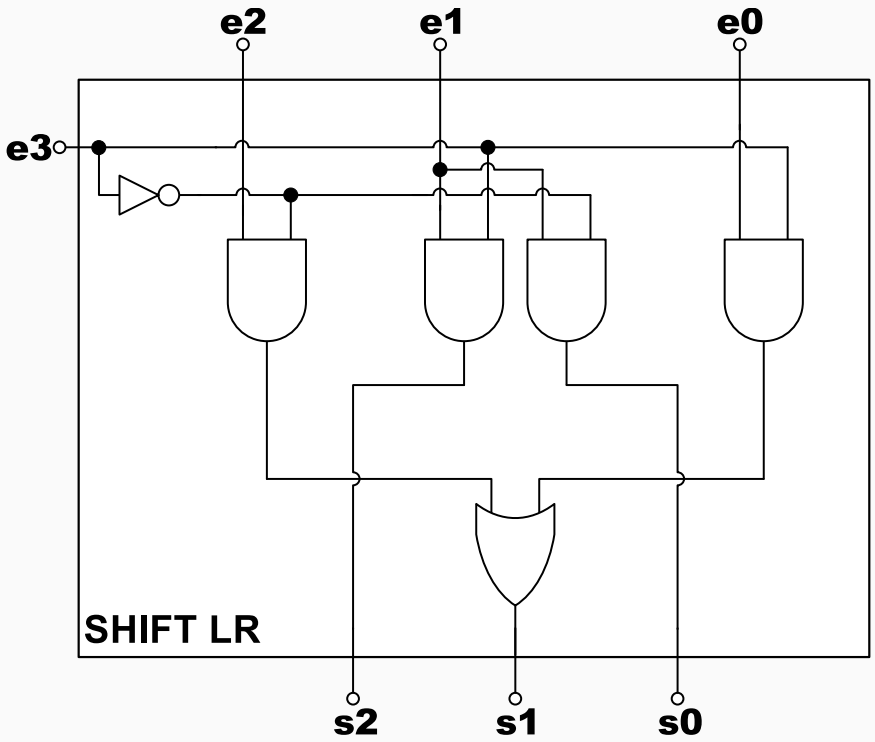




Timing

¡Las compuertas no son instantáneas!

Revisitemos nuestro Shift LR:

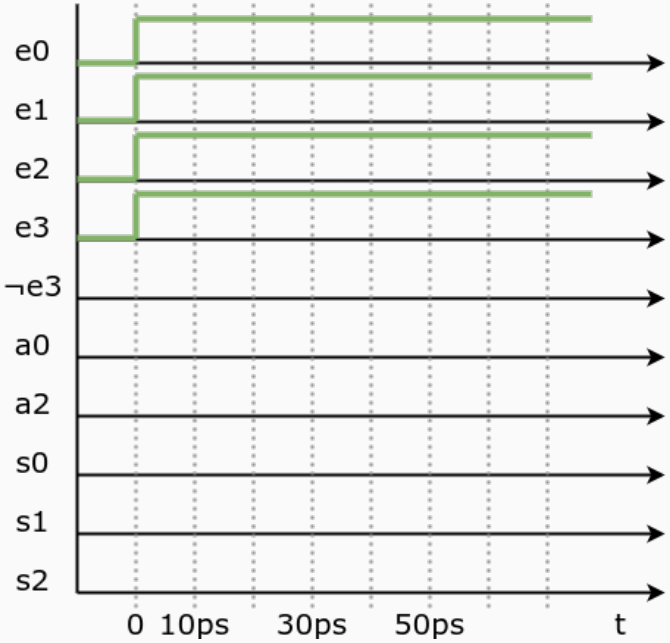


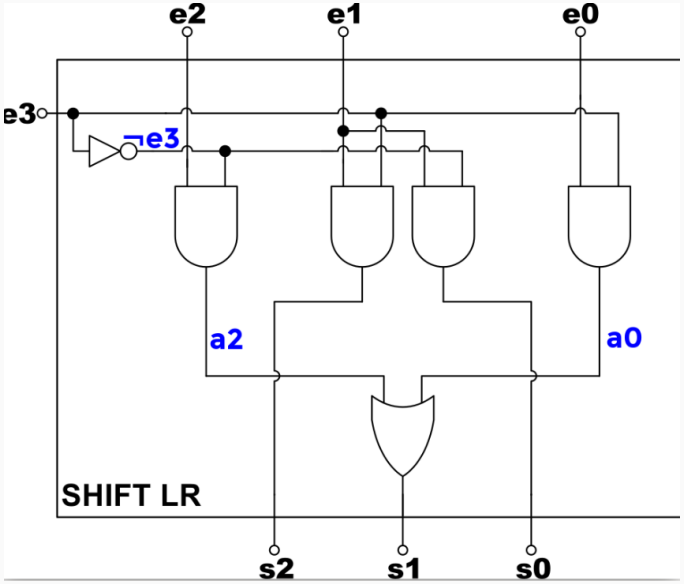
Ejercicio:

En el circuito Shift LR anterior, suponiendo (de forma optimista) que todas las compuertas tardan 10ps en poner un resultado válido en sus salidas:

- Dibujar el diagrama de tiempos para cuando todas las entradas cambian simultáneamente de '0' a '1'.
- ¿Cuánto es el mínimo tiempo que se debe esperar para leer un resultado válido de su salida?

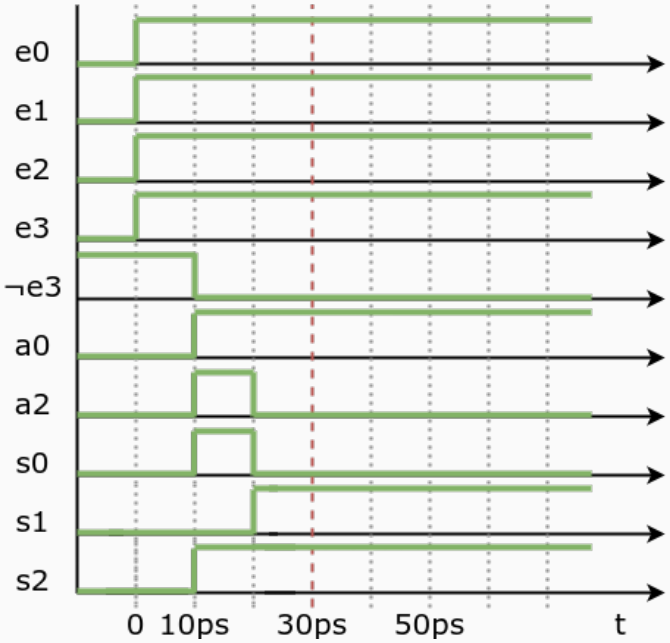
Solución: Hagamos un diagrama de tiempos[2]:

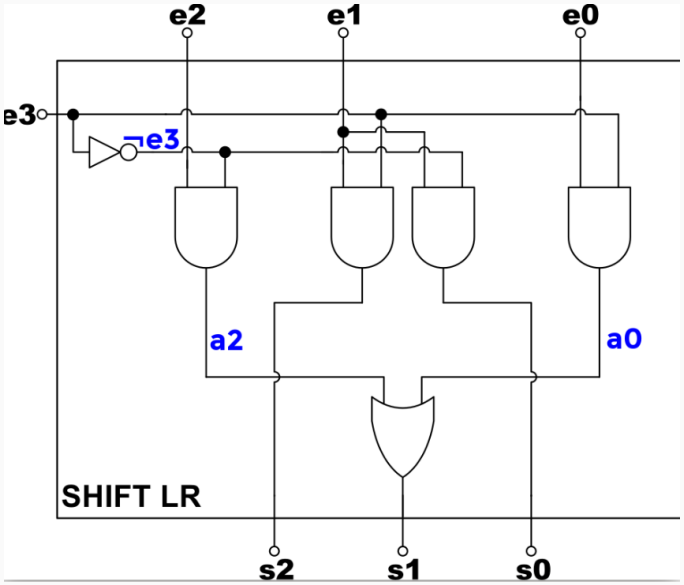




[2]Y nombremos a las señales que no tienen nombre

Solución: Diagrama de tiempos





Solución: Es interesante notar:

Solución: Es interesante notar:

- En un circuito combinatorio el tiempo que tarda la salida en estabilizarse depende de la cantidad de *capas* de compuertas (*latencia*)

Solución: Es interesante notar:

- En un circuito combinatorio el tiempo que tarda la salida en estabilizarse depende de la cantidad de *capas* de compuertas (*latencia*)
- En este caso debemos esperar al menos $3 \cdot 10ps = 30ps$ para poder leer la salida.

Solución: Es interesante notar:

- En un circuito combinatorio el tiempo que tarda la salida en estabilizarse depende de la cantidad de *capas* de compuertas (*latencia*)
- En este caso debemos esperar al menos $3 \cdot 10ps = 30ps$ para poder leer la salida.

¿Cómo enfrentamos este problema?

Secuenciales...

Conclusiones

La práctica...

- Con lo visto hoy pueden realizar la **parte A de la práctica 2**.
- Pueden usar el purpleLogisim evolution (Requiere Java 16 o superior. Para ejecutarlo, teclear en una consola `java -jar logisim-evolution-3.8.0-all.jar` desde la carpeta donde se encuentra el archivo descargado.)
- El **martes 18 de abril**(el 11 hay paro) tenemos el **primer taller** de la materia, el cual es **obligatorio**. Será en los laboratorios del pabellón **Cero+Infinito** (ver cuales en el cronograma, que está en el campus).
- Bibliografía recomendada: *The Essentials of Computer Organization and Architecture - Linda Null - Capítulo 3*