Microarquitectura 01: Sistemas combinacionales

Organización del computador - FIUBA

2.^{do} cuatrimestre de 2023

Última modificación: Sun Sep 10 16:17:47 2023 -0300

Créditos

Para armar las presentaciones del curso utilizamos:

R. E. Bryant and D. R. O'Hallaron, *Computer systems: a programmer's perspective*, Third edition, Global edition. Boston Columbus Hoboken Indianapolis New York San Francisco Cape Town: Pearson, 2015.

D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*, RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, 2017.

J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. 2017.

El contenido de los slides está basado en las presentaciones de Patricio Moreno y de Organización del Computador I - FCEN.

Tabla de contenidos

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	$\overline{A.B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}.\overline{B}$

Tarea: ¡Demostrarlas!

Ejercicio I

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

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

Solución:

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

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

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

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

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

Elementos de diseño: Implementación de operaciones booleanas

Hay muchas formas de implementar una operación booleana:

Ya las conocemos en C:

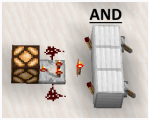
```
int a, b, s;
...
s = a & b;
```

En VHDL, un leguaje descriptor de hardware:

```
signal a, b, s: std_logic;
...
s <= a and b;
```

También las podemos representar gráficamente (por ejemplo en Logisim):

...o en *Minecraft*:



Elementos de diseño: Implementación de operaciones booleanas

Son todas las formas iguales?
NO, depende del contexto: tenemos que razonar en función del comportamiento.

$$\textit{Controlflow} \neq \textit{Dataflow}$$

Válido en ejecución secuencial, *Controlflow* (e.g. C):

```c
int a, b, s;
...
s = a & b;
s = a | b; // Ok!
```

NO válido en ejecución concurrente *Dataflow* (e.g. VHDL):

```vhdl
signal a, b, s: std_logic;
...
s <= a and b;
s <= a or b;  # Error!
```

Hoy pensaremos en ***Dataflow***: todavía no introdujimos el concepto de tiempo → **memoria**[1].

Elementos de diseño: Tablas de verdad

Son representaciones que nos permiten observar todas las salidas para todas las combinaciones de entradas[2].

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

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

Repaso de operaciones booleanas: NOT

Gráficamente:

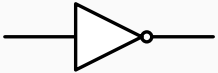


Tabla de verdad:

A	NOT A
0	1
1	0

En VHDL:

```
o <= not a;
```

Repaso de operaciones booleanas: AND

Gráficamente:

Tabla de verdad:

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

En VHDL:

```
o <= a and b;
```

Repaso de operaciones booleanas: OR

Gráficamente:

Tabla de verdad:

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

En VHDL:

```
o <= a or b;
```

Repaso de operaciones booleanas: XOR u OR-EXCLUSIVA

Gráficamente:

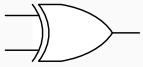


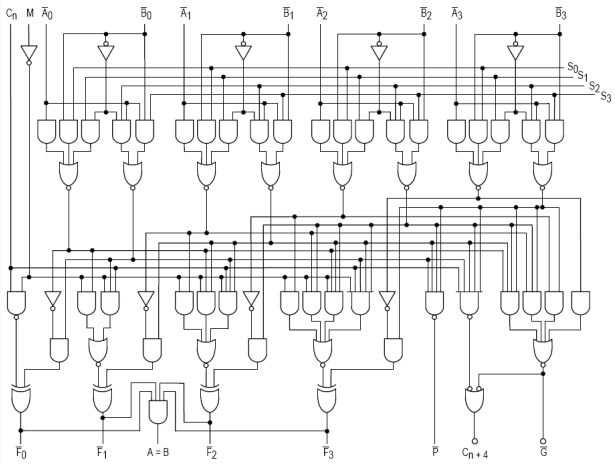
Tabla de verdad:

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

En VHDL:

```
o <= a xor b;
```

Desde el museo: ALU 74181



Conceptos importantes

- **Abstracción**:
  Un modelo del comportamiento del sistema.

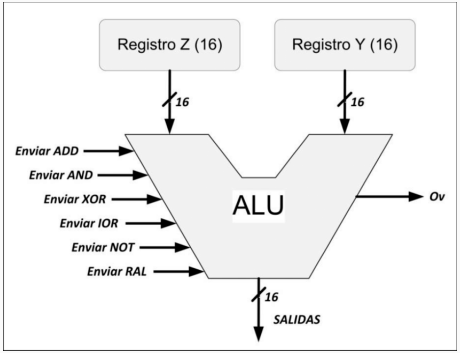
- **Encapsulamiento**:
  Crear un contenedor o capsula con una o varios componentes. Una partición del sistema.

- **Ocultamiento de la información**:
  Una vista del sistema.

¡Son conceptos **diferentes**, pero muy **relacionados**!

ALU revisitada

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



Entradas/Salidas

Establecen el sentido de la información:
En la ALU anterior se representan con las flechas...

En VHDL:

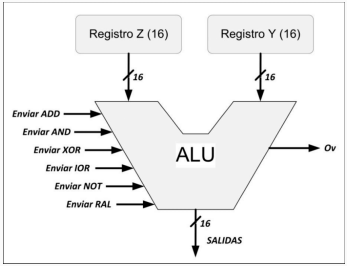
```
entity sumador_simple is
port(a: in std_logic; b:in std_logic;
     s: out std_logic);
end entity;
```

El mismo concepto se aplica al SW:

```
bool sumador_simple(bool a; bool b);
```

Entradas/Salidas: Tipos

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

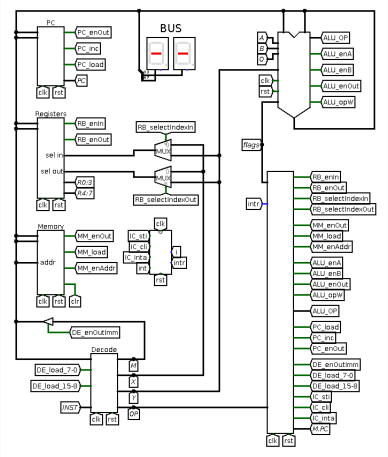


NO

Datos vs. Control

Diseño jerárquico

El micro **OrgaSmall**:



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

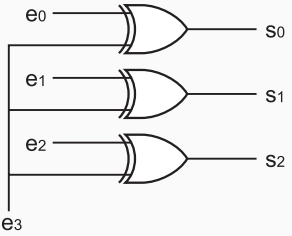
¡Divide y conquista! Primero, con un bit...

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

Como suma de productos,
$(\overline{ei} \cdot ek) + (ei \cdot \overline{ek})$

¡Oh! casualidad, es una XOR ($\oplus$)
$(\overline{A} \cdot B) + (A \cdot \overline{B}) = A \oplus B$

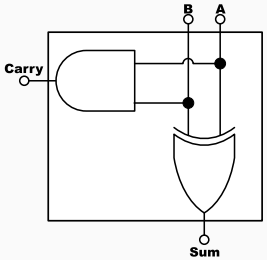
Extendiendo a 3 bits.. Solucion con XOR:



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.
Solución:

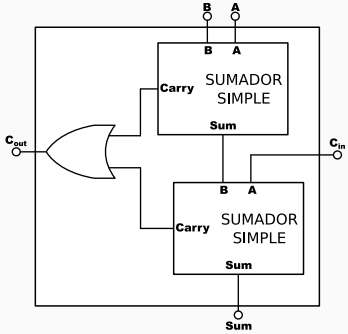
A	*B*	Sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



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. **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.

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 $s_{k-1} = 0$

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 $s_{k-1} = 0$

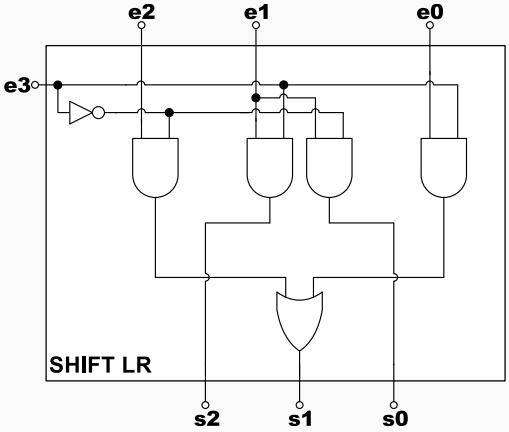
Solución:

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

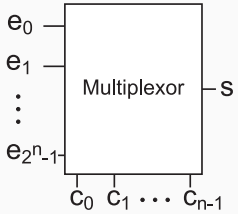
$$e_3.e_1 \qquad \overline{e_3}.e_1 \qquad e_3.e_0 + \overline{e_3}.e_2$$

Ejercicio IV - Shift

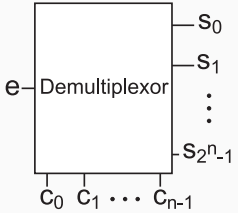
Solución:



Más combinatorios: Multiplexor y Demultiplexor



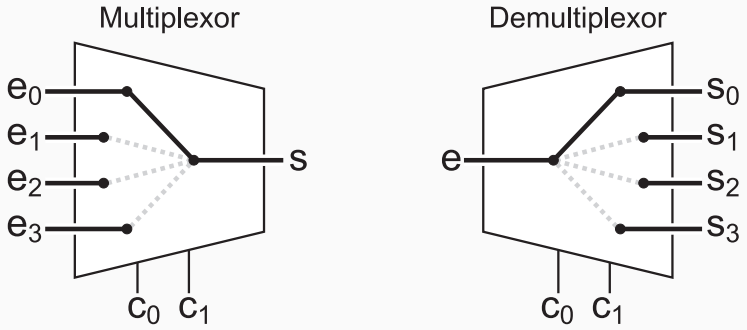
Las líneas de control *c* permiten seleccionar una de las entradas *e*, la que corresponderá a la salida *s*.



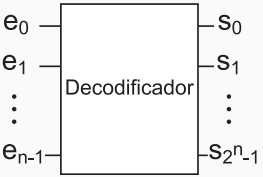
Las líneas de control *c* permiten seleccionar cual de las salidas *s* tendrá el valor de *e*.

Multiplexor y Demultiplexor

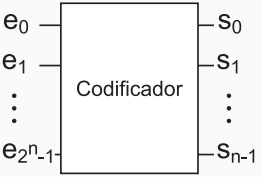
- Ejemplo,



Más combinatorios: Codificador y Decodificador



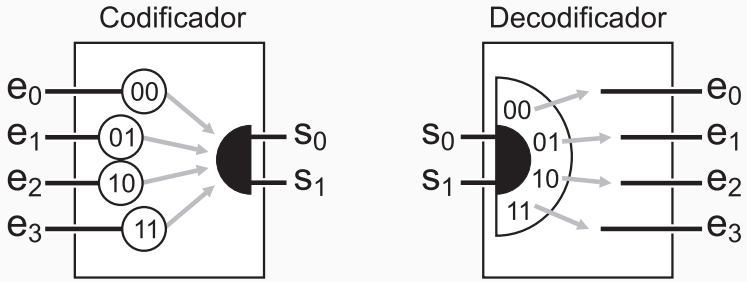
Cada combinación de las líneas *e* corresponderá a una sola línea en alto de la salida *s*.



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

Codificador y Decodificador

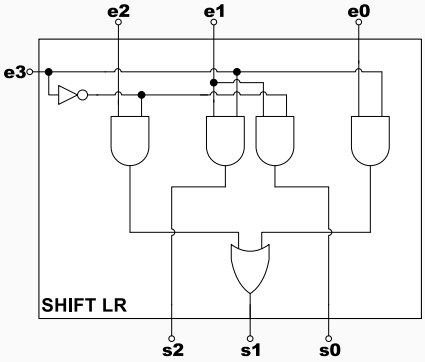
- Ejemplo,



Timing

<Las compuertas no son instantáneas!

Revisitemos nuestro Shift LR:



Timing

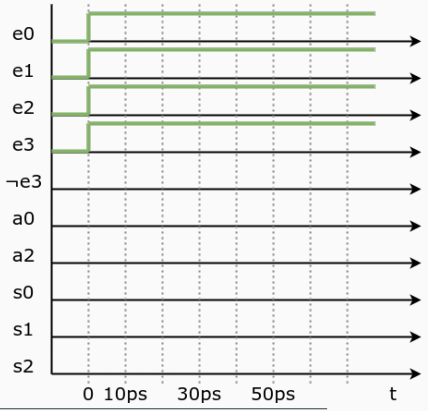
Ejercicio:

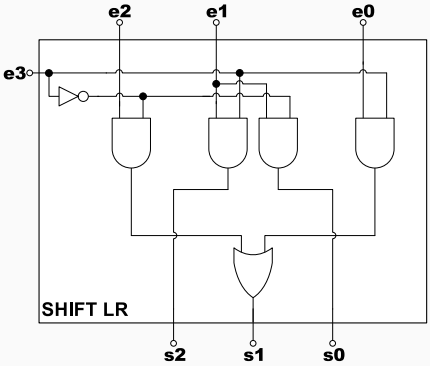
En el circuito Shift LR anterior, suponiendo (de forma optimista) que todas las compuertas tardan 10ps en poner un resultado válido en sus salidas:

- 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?

Timing

Solución: Hagamos un diagrama de tiempos[3]:

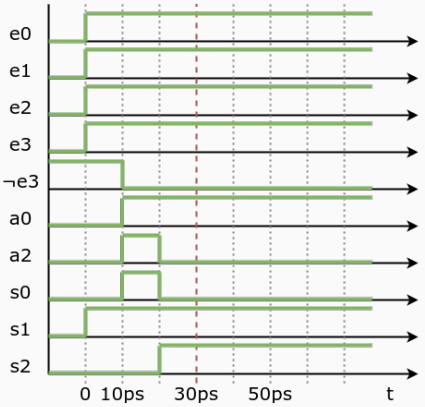


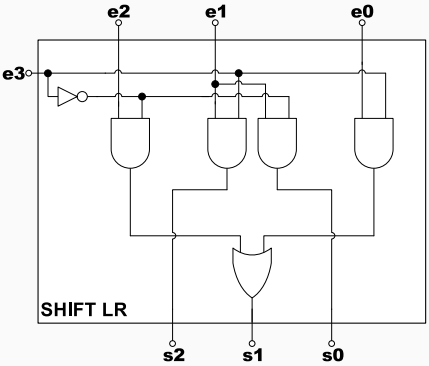


[3] Y nombremos a las señales que no tienen nombre

Timing

Solución: Diagrama de tiempos





Timing

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?

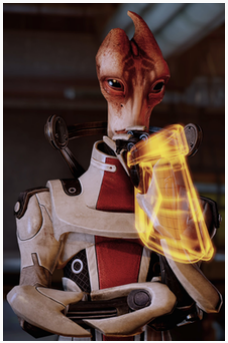
Secuenciales...

La práctica...

- Con lo visto hoy van a poder realizar el taller de la próxima práctica.
- Van a usar Logisim-evolution para probar sus sistemas.
- Para el taller:
  - Tienen que tener el Logisim-evolution instalado (pueden instalarlo del link de arriba, o del gestor de paquetes de su distro favorita).
  - Van a tener que compartirnos pantalla.
  - Como siempre: ¡Los queremos escuchar! Prueben los micrófonos de sus compus/celulares.

¡Eso es todo amigos!

¿Preguntas?



Licencia del estilo de beamer

Obtén el código de este estilo y la presentación demo en

`github.com/pamoreno/mtheme`

El estilo *en sí* está licenciado bajo la Creative Commons Attribution-ShareAlike 4.0 International License. El estilo es una modificación del creado por Matthias Vogelgesang, disponible en

`github.com/matze/mtheme`