# Clase 09 - Compuertas lógicas y lógica proposicional

## IIC1001 - Algoritmos y Sistemas Computacionales

Cristian Ruz – cruz@uc.cl

Jueves 9-Abril-2026

Departamento de Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica de Chile

# Contenidos

# Contenidos

# Contacto


🐐

vicente.cabra@uc.cl

Vicente Cabra

Ayudante jefe





🐸

alejandro.tapia@uc.cl

Alejandro Tapia

Ayudante jefe





🛼

gabriel.sandoval@uc.cl

Gabriel Sandoval

Ayudante jefe




# Contenidos

# Temas del curso

## Sistemas computacionales

- Representación datos, números y codificación
- Funcionamiento hardware, procesadores y memoria.
- Funcionamiento de sistemas operativos: ejemplo scheduling
- Funcionamiento de Internet
- Herramientas computacionales: github + latex

## Algoritmos

- Algoritmos y resolución de problemas
- Eficiencia algorítmica
- Estructuras secuenciales y ordenamiento
- Grafos y árboles

# Próximos eventos

**Jueves 9-Abril**: Actividad 3. En grupos aleatorios

- Compuertas lógicas
- Circuitos digitales simples

**Viernes 10-Abril, 14:50**: Ayudantía (opcional).

- Orientada a la Interrogación 1
- Sala por confirmar

**Martes 14-Abril, 17:30**: Interrogación 1.

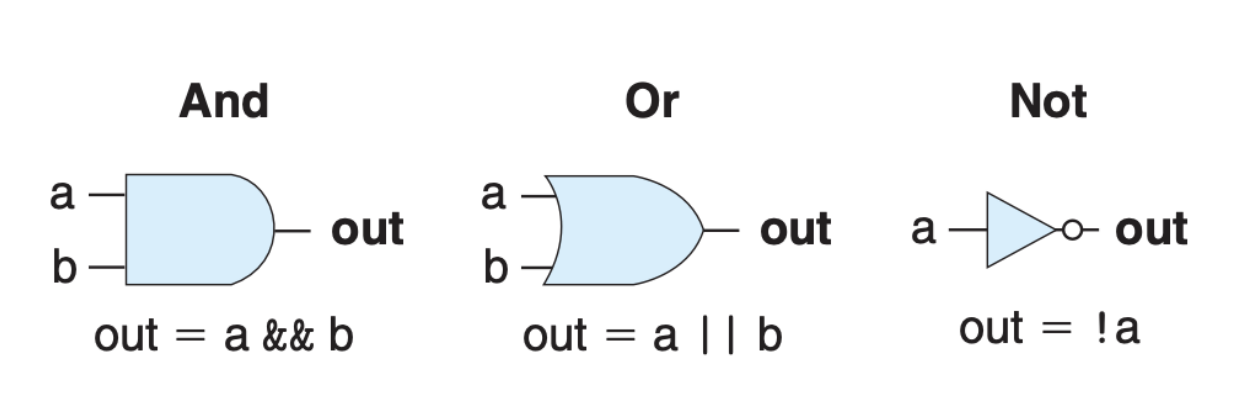
- Salas K200 y K201
- De 17:30 a 19:30
- Individual y sin apuntes.
- Hasta circuitos sumadores.
- Clase de repaso en la mañana.

# Contenidos

# Compuertas lógicas

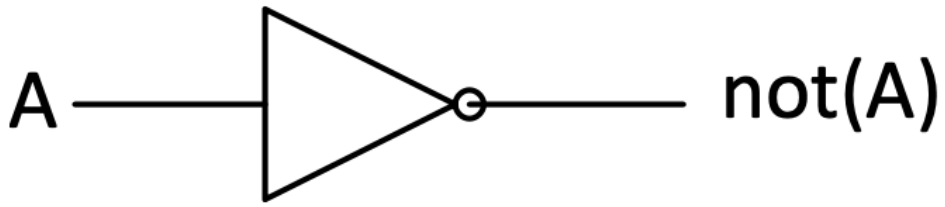
Componentes electrónicas que implementa las condiciones lógicas de Boole.

Poseen una cantidad de valores (bit) de entrada y entregan como resultado uno o más valores (bits) de salida de acuerdo a una tabla.



# Compuerta NOT

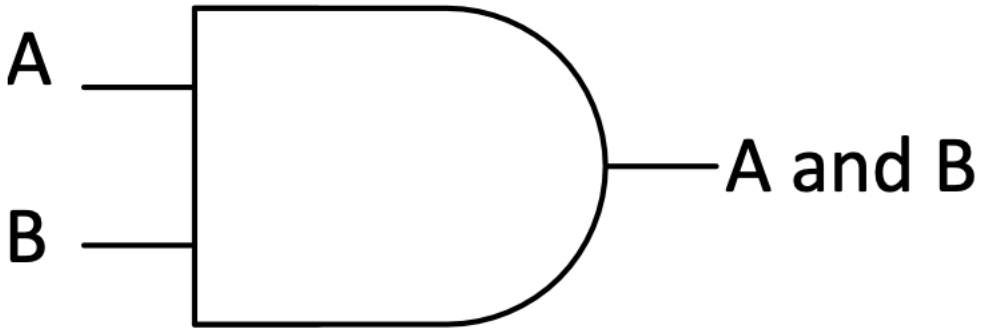
Entrada es lo opuesto de la salida: $\text{not}(A) = \neg A$




| A | not(A) |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Compuerta AND

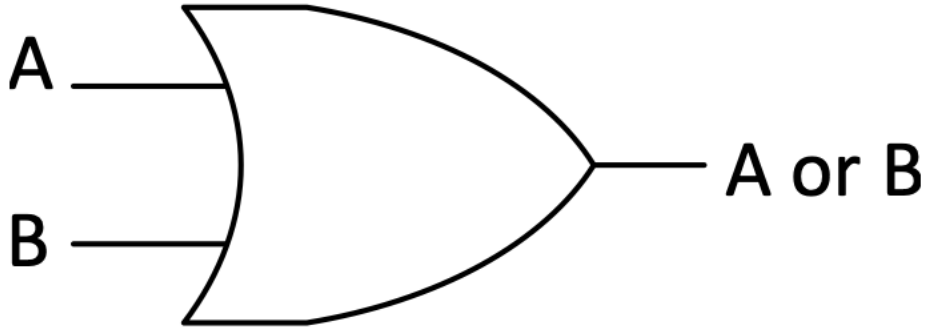
Entrada es verdadera (1) sólo cuando **ambas** entradas son verdaderas: $A$ **and** $B = A \wedge B$




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

# Compuerta OR

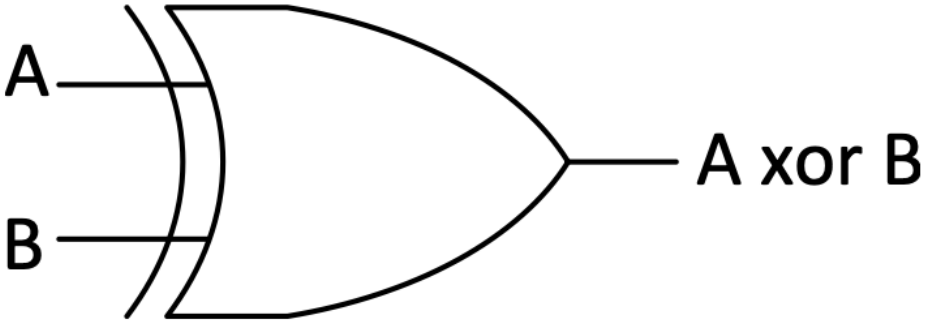
Entrada es verdadera (1) cuando **al menos una** de las entradas son verdaderas: $A \texttt{ or } B = A \vee B$




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

# Compuerta XOR

Entrada es verdadera (1) cuando **exactamente una** de las entradas son verdaderas: $A \texttt{ xor } B = A \bigoplus B$




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

# Circuitos binarios

Construir un circuito cuya salida sea 1 solamente si ambos bit de entrada son iguales.

| $a$ | $b$ | $a$ **eq** $b$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

¿Cómo se escribiría en álgebra booleana?

¿Cómo se escribiría un circuito lógico?

# Circuitos sumadores

Podemos empezar a construir circuitos que implementen operaciones aritméticas.

¿Cómo sería relación entrada/salida? $S(A, B) = A + B$

| $A$ | $B$ | $A + B$ |
|---|---|---|
| 0 | 0 | 00 |
| 0 | 1 | 01 |
| 1 | 0 | 01 |
| 1 | 1 | 10 |

# Circuitos sumadores

Podemos empezar a construir circuitos que implementen operaciones aritméticas.

¿Cómo sería relación entrada/salida? $S(A, B) = A + B$

| $A$ | $B$ | $A + B$ |
|---|---|---|
| 0 | 0 | 00 |
| 0 | 1 | 01 |
| 1 | 0 | 01 |
| 1 | 1 | 10 |

¡Pero la salida de una suma de dos números de un 1 bit son dos bit!

# Circuitos sumadores

INPUT: bit A y bit B

OUTPUT: dos bit. Podemos separarlos: $S_1$, $S_0$

| $A$ | $B$ | $A + B = S_1S_0$ | $S_1$ | $S_0$ |
|---|---|---|---|---|
| 0 | 0 | 00 | 0 | 0 |
| 0 | 1 | 01 | 0 | 1 |
| 1 | 0 | 01 | 0 | 1 |
| 1 | 1 | 10 | 1 | 0 |

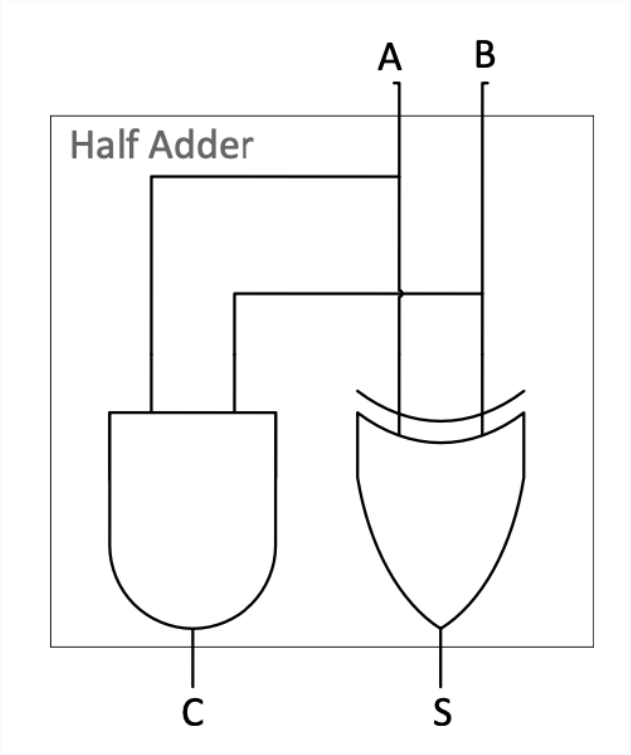
# Circuitos sumadores

INPUT: bit A y bit B

OUTPUT: dos bit. Podemos separarlos: $S_1$, $S_0$

| $A$ | $B$ | $A + B = S_1S_0$ | $S_1$ | $S_0$ |
|---|---|---|---|---|
| 0 | 0 | 00 | 0 | 0 |
| 0 | 1 | 01 | 0 | 1 |
| 1 | 0 | 01 | 0 | 1 |
| 1 | 1 | 10 | 1 | 0 |

Half-adder



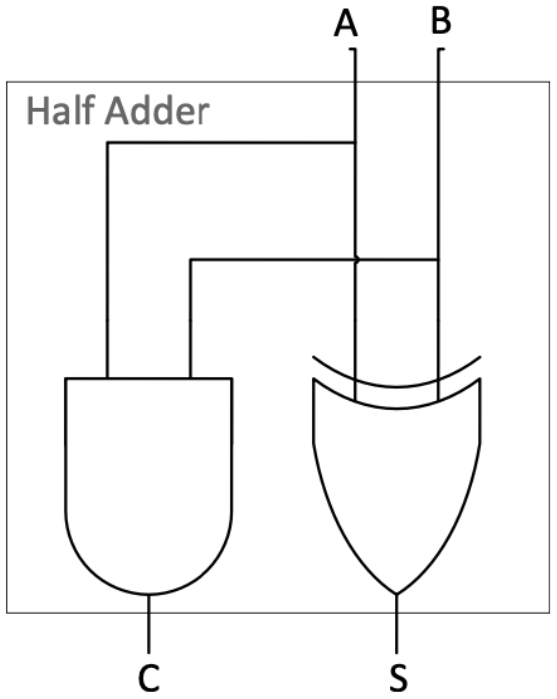
# Circuitos sumadores

INPUT: bit A y bit B

OUTPUT: dos bit. Podemos separarlos: $S_1$, $S_0$

| $A$ | $B$ | $A + B = S_1S_0$ | $S_1$ | $S_0$ |
|---|---|---|---|---|
| 0 | 0 | 00 | 0 | 0 |
| 0 | 1 | 01 | 0 | 1 |
| 1 | 0 | 01 | 0 | 1 |
| 1 | 1 | 10 | 1 | 0 |

Half-adder




Con esto podemos dos números de 1 bit.

¿Cómo sumar números más grandes?

# Circuitos sumadores

¿Cómo sumar números, por ejemplo, de 2-bit?

INPUT: dos números de dos bit: $A = A_1A_0$, $B = B_1B_0$

OUTPUT: $A + B = S$ ¿cuántos bit?

Hay que ver cómo funciona la suma

$$\begin{array}{cccccc} & & {\scriptstyle 1} & {\scriptstyle 1} & & \\ & 1 & 0 & 0 & 1 & \\ + & 0 & 0 & 1 & 1 & \\ \hline & 1 & 1 & 0 & 0 & \end{array}$$

# Circuitos sumadores

¿Cómo sumar números, por ejemplo, de 2-bit?

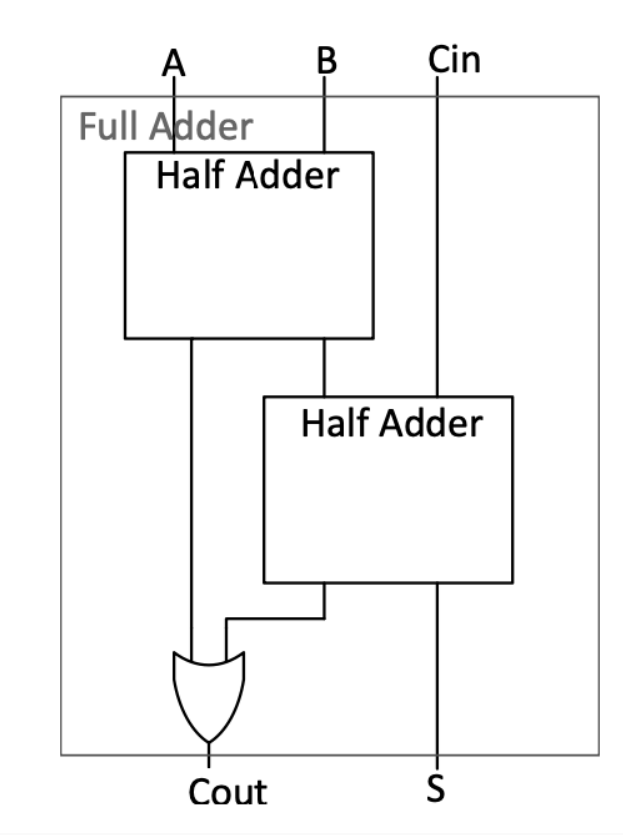
INPUT: dos números de dos bit: $A = A_1A_0$, $B = B_1B_0$

OUTPUT: $A + B = S$ ¿cuántos bit? $S = S_2S_1S_0$

Hay que ver cómo funciona la suma

$$\begin{array}{cccccc} & & {\scriptstyle 1} & {\scriptstyle 1} & & \\ & 1 & 0 & 0 & 1 & \\ + & 0 & 0 & 1 & 1 & \\ \hline & 1 & 1 & 0 & 0 & \end{array}$$

Full-adder



# Circuitos sumadores

¿Cómo sumar números, por ejemplo, de 2-bit?

INPUT: dos números de dos bit: $A = A_1A_0$, $B = B_1B_0$

OUTPUT: $A + B = S$ ¿cuántos bit?

| $A_1A_0$ | $B_1B_0$ | $S = S_2S_1S_0$ |
|---|---|---|
| 00 | 00 | 000 |
| 00 | 01 | 001 |
| 00 | 10 | 010 |
| 00 | 11 | 011 |
| 01 | 00 | 001 |
| 01 | 01 | 010 |
| 01 | 10 | 011 |
| 01 | 11 | 100 |
| 10 | 00 | 010 |
| 10 | 01 | 011 |
| 10 | 10 | 100 |
| 10 | 11 | 101 |
| 11 | 00 | 011 |
| 11 | 01 | 100 |
| 11 | 10 | 101 |
| 11 | 11 | 110 |

# Circuitos sumadores

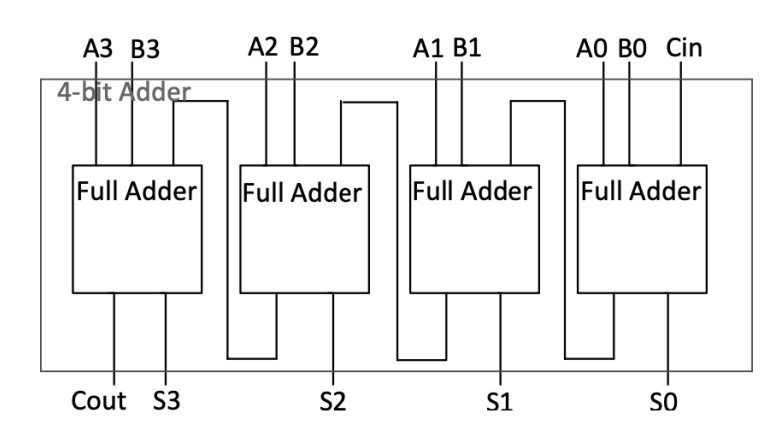
¿Cómo sumar números, por ejemplo, de 2-bit?

INPUT: dos números de dos bit: $A = A_1A_0$, $B = B_1B_0$

OUTPUT: $A + B = S$ ¿cuántos bit? $S = S_2S_1S_0$

| $A_1A_0$ | $B_1B_0$ | $S = S_2S_1S_0$ |
|---|---|---|
| 00 | 00 | 000 |
| 00 | 01 | 001 |
| 00 | 10 | 010 |
| 00 | 11 | 011 |
| 01 | 00 | 001 |
| 01 | 01 | 010 |
| 01 | 10 | 011 |
| 01 | 11 | 100 |
| 10 | 00 | 010 |
| 10 | 01 | 011 |
| 10 | 10 | 100 |
| 10 | 11 | 101 |
| 11 | 00 | 011 |
| 11 | 01 | 100 |
| 11 | 10 | 101 |
| 11 | 11 | 110 |

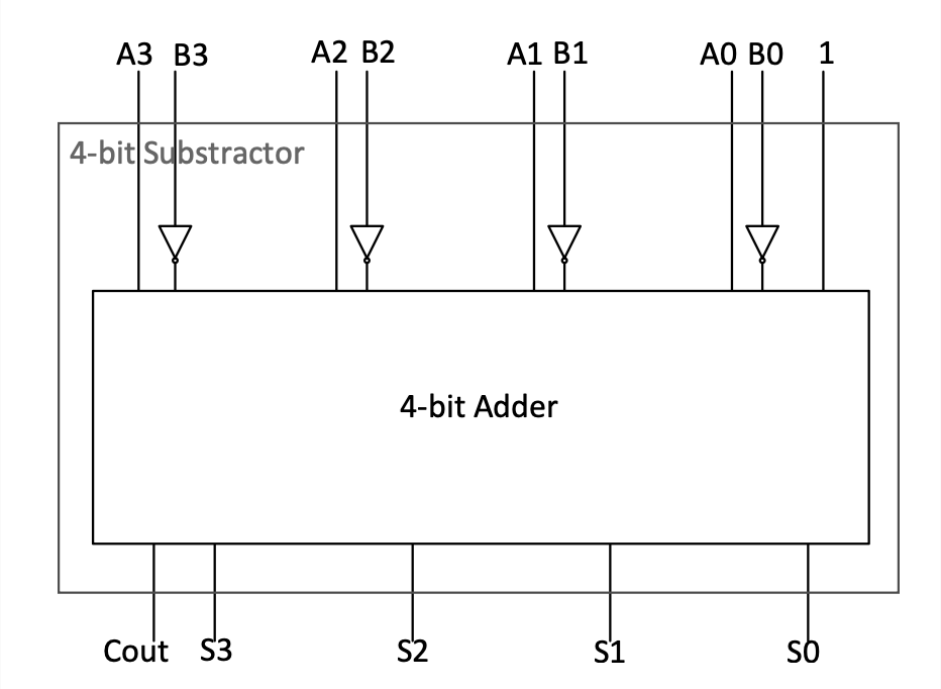
Full-adder



# Circuitos sumadores

También se puede implementar restadores en complemento de 2.

Conversión a complemento de 2 es: (1) Invertir los bits, (2) Sumar 1

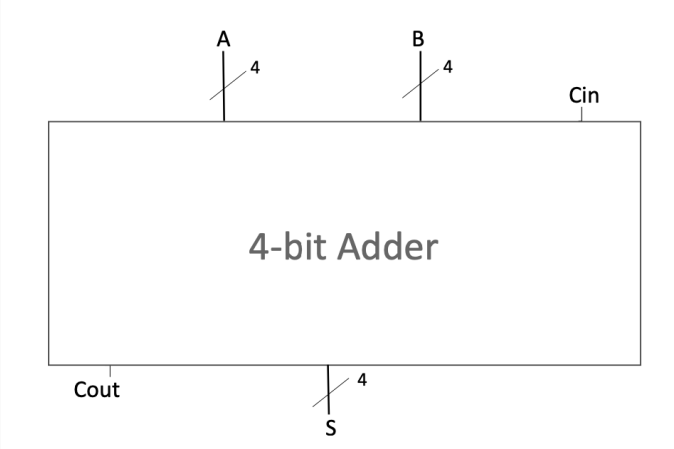


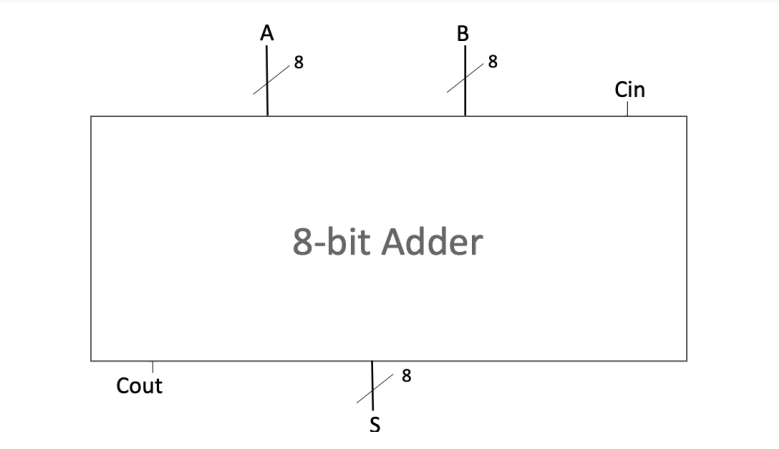

Restador de 4 bits

# Abstrayendo componentes

Al construir circuitos más complejos podemos ocultar detalles.

La notación se simplifica indicando cuántas líneas (bit) lleva cada entrada. Estas líneas se conoce como ”buses”.

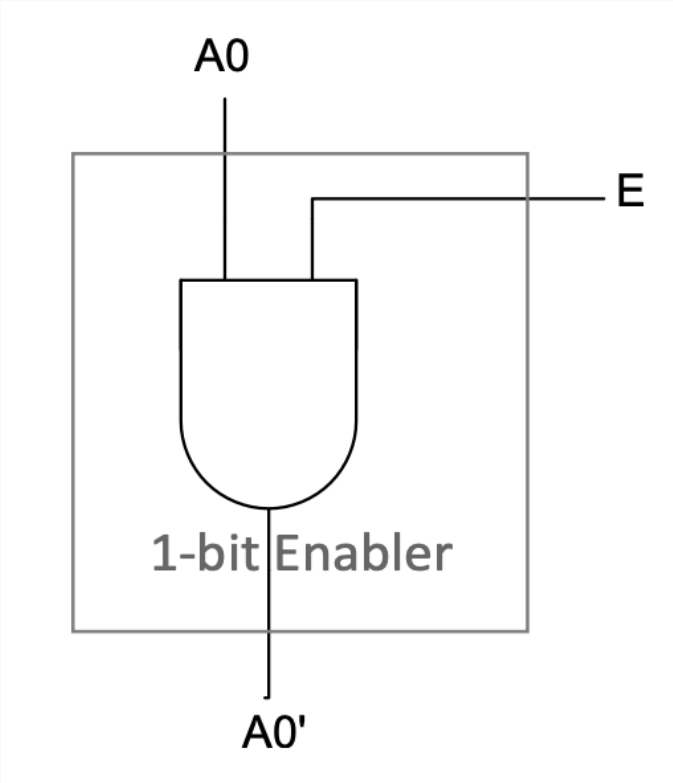


# Otros componentes: Enabler

Componente que permite habilitar o deshabilitar una salida

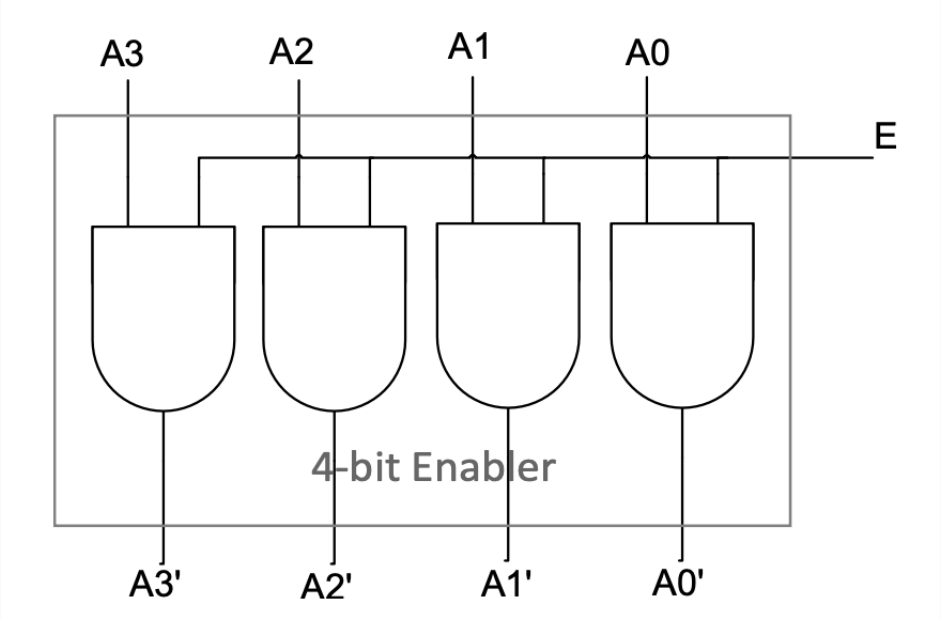
| E | A0’ |
|---|---|
| 0 | 0 |
| 1 | A0 |




Enabler de 1 bit

# Enabler

Se puede implementar para circuitos con más bit de entrada




Enabler de 4 bits

# Ejemplo 1

Calcular la salida del circuito:

$$Y = (A \wedge B) \vee \neg A$$

# Ejemplo 1

Calcular la salida del circuito:

$$Y = (A \wedge B) \vee \neg A$$

**Tabla de verdad:**

| A | B | $A \wedge B$ | $\neg A$ | Y |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 |

# Ejercicio 1

Construir la tabla de verdad para:

$$Y = (A \vee B) \wedge \neg B$$

# Ejercicio 1

Construir la tabla de verdad para:

$$Y = (A \vee B) \wedge \neg B$$

**Pasos:**

- Calcular $A \vee B$
- Calcular $\neg B$
- Calcular AND de resultados

# Sumadores

Los sumadores permiten sumar números binarios.

Tipos:

- Medio sumador (Half Adder)
- Sumador completo (Full Adder)

# Medio Sumador

Entradas: A, B

Salidas:

- Suma: $S = A \oplus B$
- *Carry*: $C = A \wedge B$

| A | B | S | C |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

# Sumador Completo

Entradas: A, B, $C_{in}$

Salidas:

- $S = A \oplus B \oplus C_{in}$
- $C_{out} = AB \wedge C_{in}(A \oplus B)$

# Tabla Sumador Completo

| A | B | $C_{in}$ | 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 |

# Ejemplo 2

Sumar:

$$1 + 1$$

# Ejemplo 2

Sumar:

$$1 + 1$$

- Suma = 0
- Acarreo = 1

Resultado binario: 10

# Ejercicio 2

Diseñar un circuito que calcule:

$$S = A \oplus B \oplus C$$

# Ejercicio 2

Diseñar un circuito que calcule:

$$S = A \oplus B \oplus C$$

**Preguntas:**

- ¿Cuántas compuertas XOR necesitas?
- ¿Cómo conectarías las entradas?

# Ejercicio 3

Construir un circuito que:

- Encienda una salida si exactamente una entrada es 1.

# Ejercicio 3

Construir un circuito que:

- Encienda una salida si exactamente una entrada es 1.

**Pista:**

- Usa XOR

# Conclusión

- Las compuertas lógicas son la base de los circuitos digitales.
- Los sumadores permiten operaciones aritméticas.
- Las tablas de verdad ayudan a entender el comportamiento.

# Ejercicio Simple

Dada la función:

$$Y = \neg A \wedge B \vee A \wedge \neg B$$

# Ejercicio Simple

Dada la función:

$$Y = \neg A \wedge B \vee A \wedge \neg B$$

- Construya la tabla de resultados
- ¿Qué compuerta representa esta función?

# Ejercicio Simple

Dada la función:

$$Y = \neg A \wedge B \vee A \wedge \neg B$$

- Construya la tabla de resultados
- ¿Qué compuerta representa esta función?

**Solución:**

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

# Ejercicio Intermedio

Dada la función:

$$Y = (A \vee B) \wedge (\neg A \vee C)$$

# Ejercicio Intermedio

Dada la función:

$$Y = (A \vee B) \wedge (\neg A \vee C)$$

- Construir tabla de resultados
- ¿Es posible simplificarlo usando álgebra booleana?
- Diseñar circuito que lo implemente

# Ejercicio Intermedio

Dada la función:

$$Y = (A \vee B) \wedge (\neg A \vee C)$$

- Construir tabla de resultados
- ¿Es posible simplificarlo usando álgebra booleana?
- Diseñar circuito que lo implemente

$$Y = A \wedge \neg A + A \wedge C + B \wedge \neg A + B \wedge C$$

$$A \wedge \neg A = 0$$

$$Y = A \wedge C + B \wedge \neg A + B \wedge C$$

# Ejercicio Desafiante

Diseñar un circuito con entradas A, B, C tal que:

- La salida sea 1 si al menos dos entradas son 1

# Ejercicio Desafiante

Diseñar un circuito con entradas A, B, C tal que:

- La salida sea 1 si al menos dos entradas son 1

**Tareas:**

- Construir tabla de verdad
- Obtener función booleana
- Dibujar circuito

# Ejercicio Desafiante

Diseñar un circuito con entradas A, B, C tal que:

- La salida sea 1 si al menos dos entradas son 1

**Tareas:**

- Construir tabla de verdad
- Obtener función booleana
- Dibujar circuito

**Solución:**

$$Y = A \wedge B + A \wedge C + B \wedge C$$

- Tres compuertas AND
- Una compuerta OR de 3 entradas

# Comparador de 1 bit

Entradas: A, B

Salidas:

- $A > B$
- $A = B$
- $A < B$

# Comparador de 1 bit

Entradas: A, B

Salidas:

- $A > B$
- $A = B$
- $A < B$

**Funciones:**

$$A > B = A \cdot \neg B$$

# Comparador de 1 bit

Entradas: A, B

Salidas:

- $A > B$
- $A = B$
- $A < B$

**Funciones:**

$$A > B = A \cdot \neg B$$

$$A < B = \neg A \cdot B$$

# Comparador de 1 bit

Entradas: A, B

Salidas:

- $A > B$
- $A = B$
- $A < B$

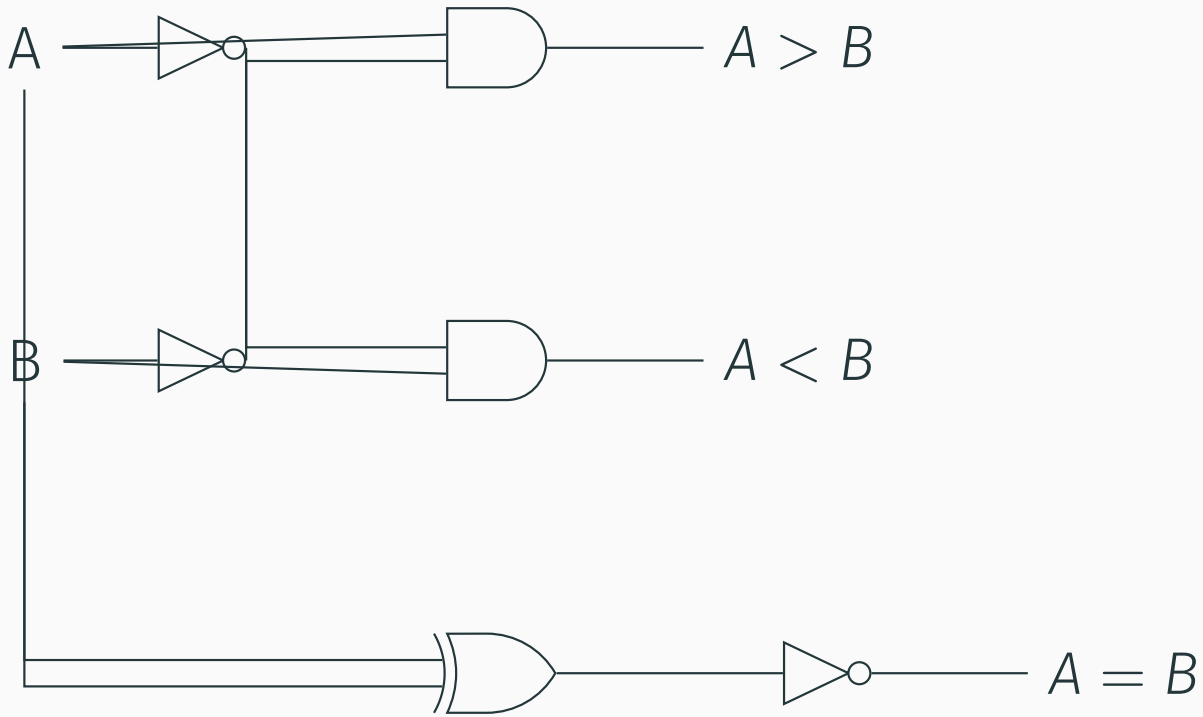
**Funciones:**

$$A > B = A \cdot \neg B$$

$$A < B = \neg A \cdot B$$

$$A = B = \neg(A \oplus B)$$

# Circuito Comparador (TikZ)



# Ejercicio Comparador

Diseñar un comparador de 2 bits:

$$A = A_1A_0, \quad B = B_1B_0$$

# Ejercicio Comparador

Diseñar un comparador de 2 bits:

$$A = A_1A_0, \quad B = B_1B_0$$

**Preguntas:**

- ¿Cuándo $A > B$?
- Expresar en lógica booleana

# Ejercicio Comparador

Diseñar un comparador de 2 bits:

$$A = A_1A_0, \quad B = B_1B_0$$

**Preguntas:**

- ¿Cuándo $A > B$?
- Expresar en lógica booleana

**Idea clave:**

$$A > B = (A_1 > B_1) + (A_1 = B_1)(A_0 > B_0)$$

- Combina comparadores de 1 bit