Laboratorio de sistemas digitales

ELO 212

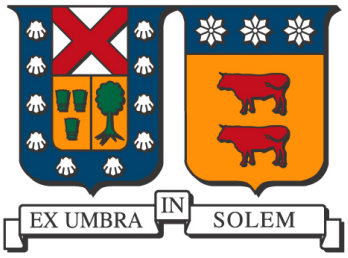
Experiencia 0

Integrantes:

Juan Pablo Sánchez

Julio López

José Catalán



Universidad Técnica Federico Santa María

2021-1

1 Diseño de un circuito digital para un full-adder

A partir de la tabla de verdad, diseñe un circuito que implemente un full-adder, utilizando solo compuertas **NOT, AND Y OR**. Muestre el esquemático resultante.

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

Table 1: Tabla de verdad.

En base a esa tabla de verdad se pueden obtener las expresiones de S y de C_{out} utilizando minterms. Así quedan expresadas tal y como se ven en las ecuaciones 1 y 2.

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C_{in}} + A\bar{B}\bar{C_{in}} + ABC \tag{1}$$

$$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C_{in}} + ABC \tag{2}$$

Esto último puede simplificarse haciendo uso de mapas de Karnaugh, obteniéndose finalmente las siguientes expresiones, las ecuaciones 3 y 4 .

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C_{in}} + A\bar{B}\bar{C_{in}} + ABC \tag{3}$$

$$C_{out} = \bar{A}B + AC_{in} + BC_{in} \tag{4}$$

Y de esta manera se puede obtener un circuito que describa las propiedades de S y C_{out} del **full adder**.

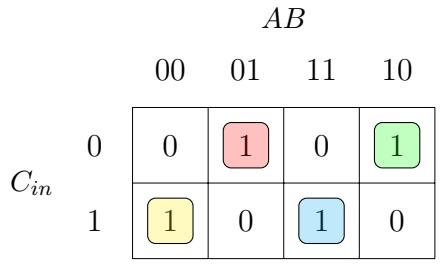


Figure 1: Mapa Karnaugh asociado a la salida S.

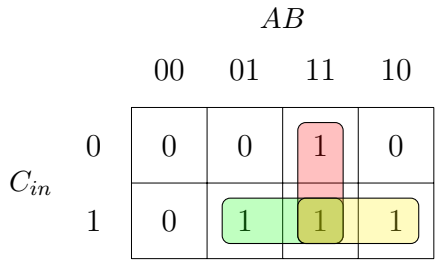


Figure 2: Mapa de Karnaugh asociado a la salida C_{out}.

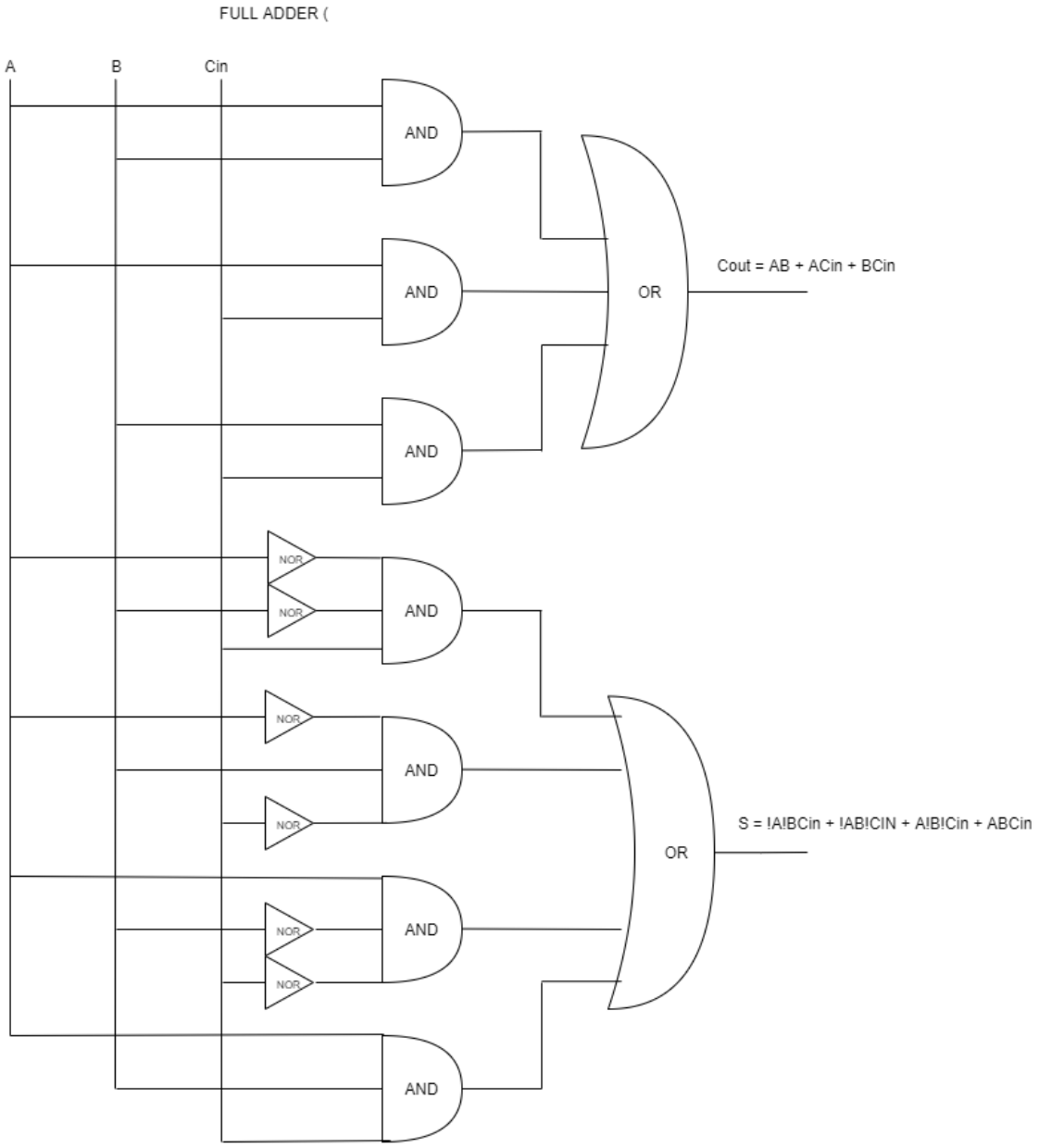


Figure 3: Diseño de Full Adder utilizando min terms y mapas de Karnaugh.

Otra manera de lograr un **full adder** es combinando dos **half adder**, estos son más fáciles de diseñar y por ende pueden obtenerse de la siguiente manera.

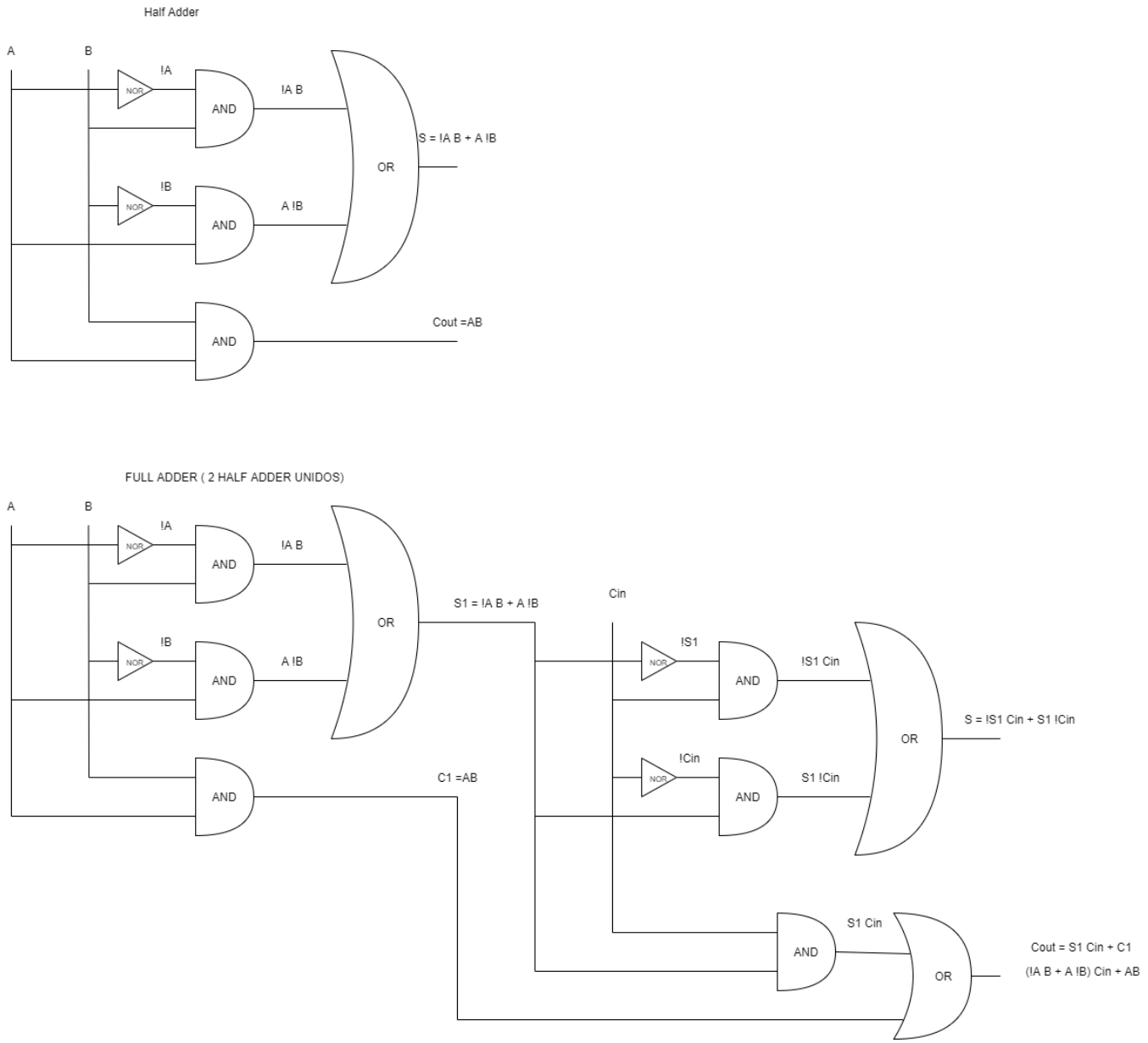


Figure 4: Diseño de Full Adder utilizando 2 Half Adder.

2 Diagrama temporal para un full-adder

En base al esquemático de la sección anterior, complete el diagrama temporal para las señales de entrada dadas.

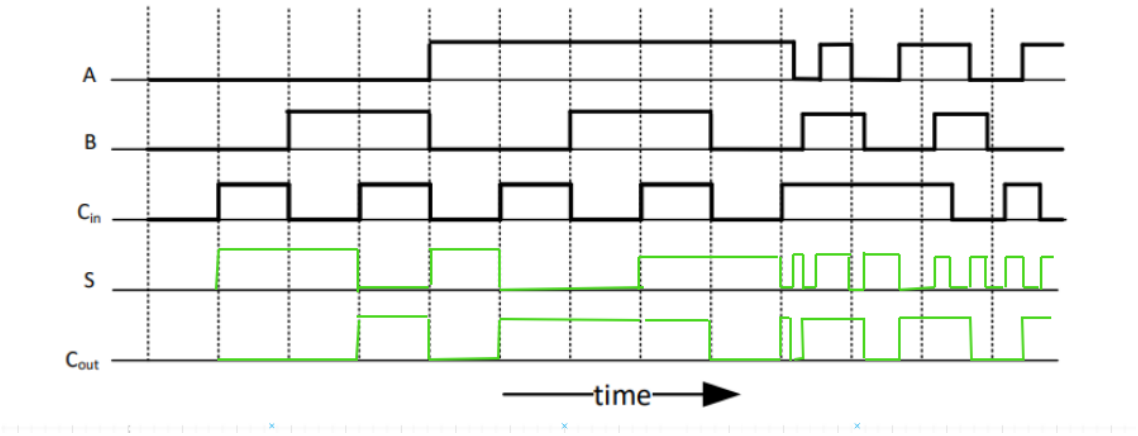


Figure 5: Timeline

3 Extensión a sumador de múltiples bits

Describir la tabla de verdad de un sumador que permita realizar la suma de dos números de **2 bits** y obtenga el esquemático correspondiente utilizando compuertas **AND, OR y NOT**. Analice la tabla de verdad y el esquemático correspondiente para sumadores de **3 y 4 bits**.

Para conseguir un sumador de **2 bits**, son necesarias 4 entradas (2 bit por cada número) y 3 salidas, las cuales corresponden a la cantidad de bits del número más grande que se puede conseguir al sumar 2 números de 2 bits. Esto puede ser representado por la siguiente tabla de verdad.

A_0	A_1	B_0	B_1	S_0	S_1	S_2
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	1	0
0	0	1	1	1	1	0
0	1	0	0	0	1	0
0	1	0	1	1	1	0
0	1	1	0	0	0	1
0	1	1	1	1	0	1
1	0	0	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	1	0
1	1	0	1	1	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	1

Table 2: Tabla de verdad.

Con esta tabla de verdad, y considerando los esquemáticos expuestos en la sección 1, podremos conseguir un sumador de 2 bits utilizando un **half-adder y full-adder** en cascada. consiguiendo el bit menos significativo con un **half-adder** (ya que para este no es necesario un **carry in**). El esquemático completo, quedaría de la siguiente manera.

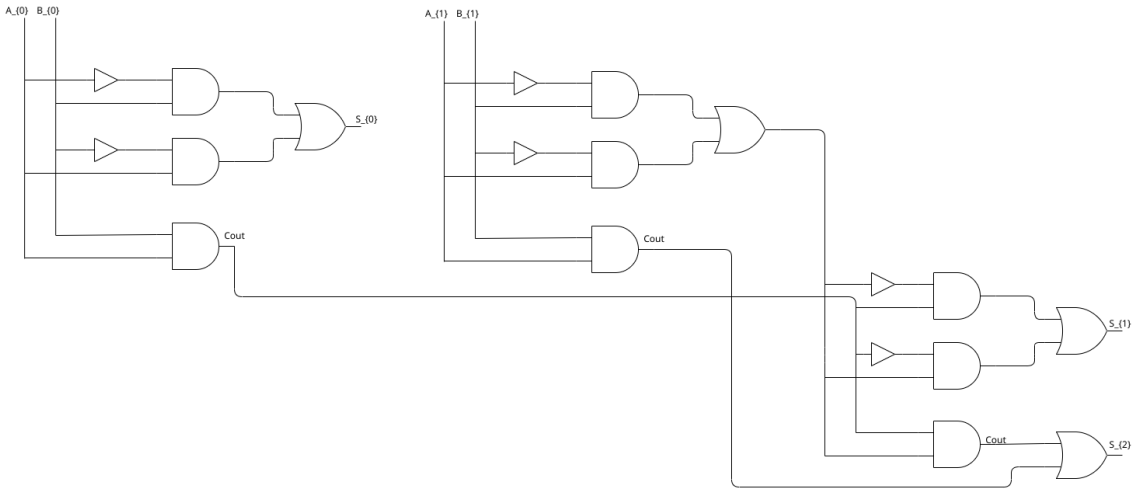


Figure 6: 2 bit adder

El procedimiento para conseguir un sumador de **3 y 4 bits**, sería seguir añadiendo **full adder** en cascada, utilizando el **carry out bit** del full adder anterior.

- ¿ Cómo escala la complejidad del diseño ?

  La complejidad del diseño escala de manera de manera lineal ya que es necesario añadir **full adder's** en cascada al esquemático propuesto. Cada **full adder** se encargará de procesar un par de bits de los números a sumar, considerando también el **carry out** anterior.

- ¿ Qué pasaría si se requiere diseñar un sumador de dos números de 32 bits ?

  Se necesitarían 31 **full adder's** en cascada para computar la suma, esto puede no ser óptimo, ya que es necesario la propagación de la señal a través de todos los **full adders** antes de tener el resultado buscado, lo que generaría demasiado delay.

- ¿ Cómo hacer más eficiente el diseño de sumadores de varios bits ?

  La forma más óptima correspondería mantener una serie de n-sumadores para poder operar con dos números con n-bits. Donde a cada sumador le ingrese un par de bits de cada número y el carry out de cada sumador ingrese al siguiente en secuencia para poder manejar correctamente la sumatoria de los bits.

4 The Nand Game

A partir de los niveles completados en el NAND Game, se apreció que los componentes fundamentales que permiten operaciones aritméticas para conformar sistemas más complejos pueden ser construidos con el uso exclusivo de compuertas NAND.

Al comparar más en detalle nuestras soluciones con respecto a las del juego, determinamos que ocurrían dos situaciones comunes. La primera de ellas era obtener una solución óptima diseñando el componente con la menor cantidad de compuertas NAND posibles, mientras que la segunda opción consistía en obtener la solución más simple con el menor número de compuertas pero no necesariamente con el menor número de NAND.

Un ejemplo de ello era el caso el diseño de un **full adder**. En nuestro caso fue construido a partir de dos **adder** más una compuerta OR dando un total equivalente a haber ocupado 13 NAND. Si bien el juego nos indicaba que era la solución más sencilla con el menor número de componentes, también era posible el poder construir la misma funcionalidad de manera más óptima ocupando solo compuertas NAND y un menor número de estas.

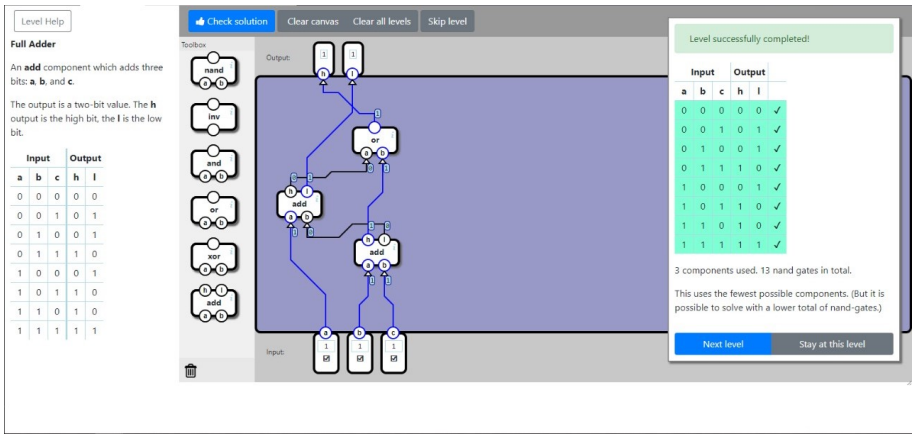


Figure 7: Solución Full Adder en NAND Game

Otro ejemplo de esta situación es al trabajar con una compuerta lógica **XOR**. Que puede ser construida de forma óptima a partir del uso exclusivo de NAND o mediante el uso de diferentes componentes. Al observar su función lógica es posible deducir las compuertas necesarias para su construcción (OR, AND y NOT) sin embargo es posible obtener la misma expresión ocupando exclusivamente compuertas NAND.

$$S = A\bar{B} + \bar{A}B \tag{5}$$

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: Tabla de Verdad XOR

Claramente observando la expresión 5 y la tabla 3 se aprecia el funcionamiento de un XOR donde si las entradas de la compuerta son iguales, se obtiene como salida un 0 o LOW mientras que si son diferentes valores de voltajes se obtiene como salida un 1 o HIGH.

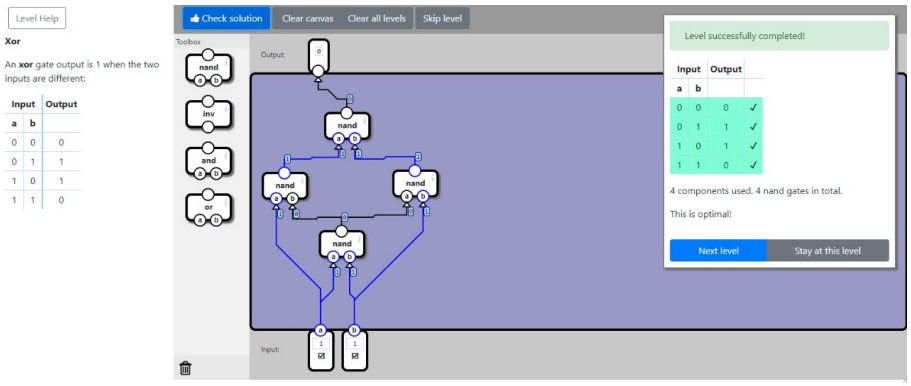


Figure 8: Solución XOR en NAND Game

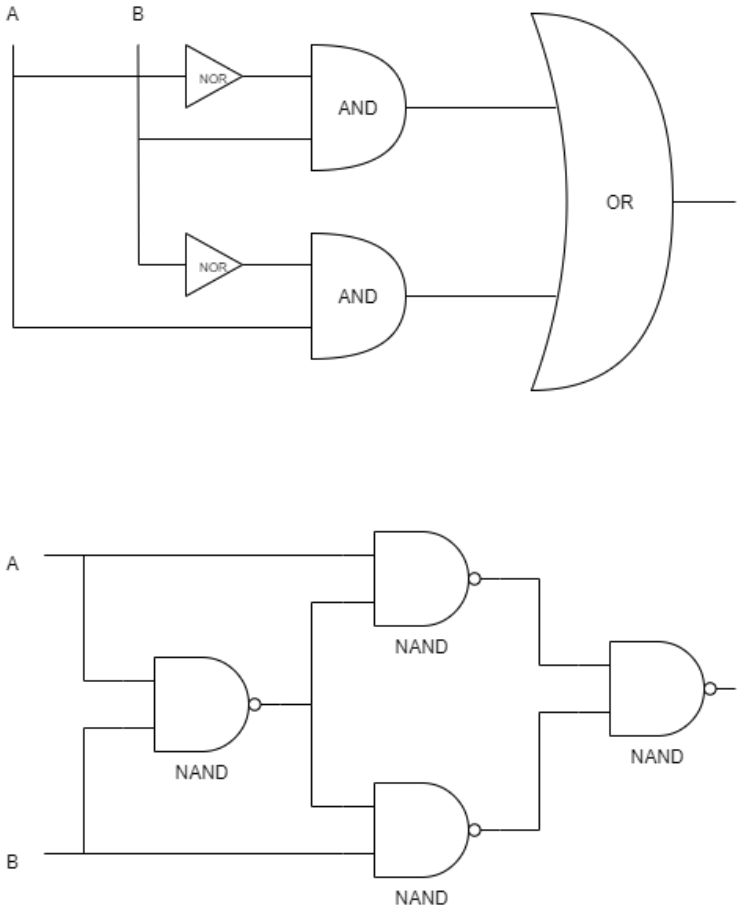


Figure 9: XOR Gate

5 Primitivas Lógicas

5.1 Multiplexor

Un multiplexor o MUX es un tipo de circuito combinacional que posee múltiples entradas y una sola salida. Este a su vez posee un selector S_e que permite usarlo como un switch, que dependiendo de su estado, transmite solo una de las entradas hacia la salida.

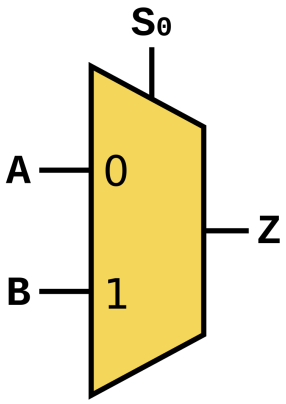


Figure 10: MUX Gate

S_0	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 11: Tabla de verdad MUX

5.2 Look Up Tables

Las Look Up Tables (LUT) son uno de los bloques fundamentales, junto a los FF, que conforman una *FPGA*. Las LUT corresponden a tablas personalizadas que almacenan las salidas (output) para una serie de entradas (input) dados y que son cargadas en memoria al momendo de encender la placa. Estás son implementadas mediante una combinación de MUX y bits en la memoria SRAM (Static RAM).

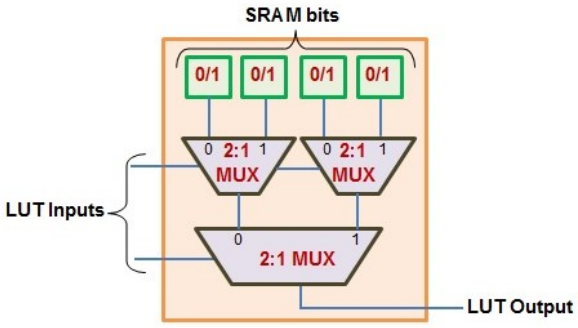


Figure 12: Estructura interna de un LUT 4-bits

La forma en que las *FPGAs* implementan lógica combinacional es mediante el uso de LUTs, cuando el dispositivo es configurado este completa las salidas de la tabla, llamadas **LUT-Mask** que se almacenan dentro de bits de la SRAM. Esto implica que una *FPGA* puede implementar cualquier función lógica ya que dependiendo de los inputs entregados el bloque de memoria apropiado será transmitido al output del LUT. Esto porque el usuario entrega las entradas que sirven como selectores S_e de los MUXs que conforman la tabla.

5.3 Flip Flop

Corresponden a dispositivos biestables (de dos estados), que sirven como dispositivos con memoria que almacenan estados para desarrollar operaciones lógicas secuenciales. Se caracterizan principalmente por:

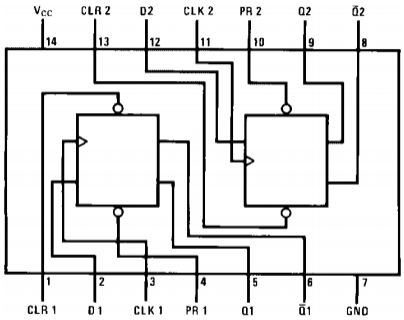
- Asumen solo uno de dos posibles estados de salida.
- Poseer un par de entradas complementarias una de otra (Q y .
- Poseer una o más entradas que pueden causar cambios en el estado del FF.

Los FF pueden ser clasificados en dos tipos:

- Sincrónicos : Las salidas del dispositivos dependen tanto de los inputs como del estado de un Clock CLK.
- Asincrónicos : Las salidas del dispositivo dependen unicamente de los inputs.

Al igual que las LUTs, los FF son elementos fundamentales dentro de la construcción y funcionamiento de una *FPGA*, en particular los FF tipo D que corresponden a un tipo de Flip Flop Síncrono que compone dichos dispositivos y que modifica sus estados de salida a partir de las ocurrencias de cantos de subida en la señal de sincronía del CLK del FF.

Connection Diagram



Function Table

Inputs				Outputs	
PR	CLR	CLK	D	Q	$\bar{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H (Note 1)	H (Note 1)
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\bar{Q}_0$

H = HIGH Logic Level
X = Either LOW or HIGH Logic Level
L = LOW Logic Level
↑ = Positive-going transition of the clock.
Q_0 = The output logic level of Q before the indicated input conditions were established.

Note 1: This configuration is nonstable; that is, it will not persist when either the preset and/or clear inputs return to their inactive (HIGH) level.

Figure 13: Diagrama de conexiónes de un circuito integrado DM7474
Tabla de Verdad de un FF Tipo D