



## **IMPLEMENTACIÓN DE UNA ALU EN UNA FPGA**

**CONVOCATORIA: JUNIO 2018**

**ALUMNA: MARÍA ISABEL DÍAZ GALIANO**

**TUTOR: JOSÉ JESÚS GARCÍA RUEDA**

**GRADO: INGENIERÍA EN DESARROLLO DE CONTENIDOS DIGITALES**



# ÍNDICE

1. Introducción
  - 1.1.**Motivación
  - 1.2.**Objetivo
  - 1.3.**Visión general del proceso de desarrollo del proyecto
2. Estado del arte
  - 2.1.**Circuitos digitales
  - 2.2.**Lógica programable
  - 2.3.***ALU (Arithmetic Logic Unit)*
    - 2.3.1. Historia
    - 2.3.2. Funcionamiento general
    - 2.3.3. Operaciones
      - 2.3.3.1. Operaciones aritméticas
      - 2.3.3.2. Operaciones lógicas
      - 2.3.3.3. Operaciones de desplazamiento
  - 2.4.**Concepto básico de la matriz AND
  - 2.5.**Tecnologías de proceso basadas en conexiones programables
    - 2.5.1. Tecnología basada en fusible
    - 2.5.2. Tecnología basada en antifusible
    - 2.5.3. Tecnología basada en EPROM (*Electrically Programmable Read-Only Memory*)
    - 2.5.4. Tecnología basada en EEPROM (*Electrically Erasable Programmable Read-Only Memory*)
    - 2.5.5. Tecnología basada en SRAM (*Static Random Access Memory*)
  - 2.6.**Tipos de dispositivos lógicos programables (Programmable Logic Devices, PLDs)
    - 2.6.1. SPLD (Simple Programmable Logic Device)
      - 2.6.1.1. PAL (Programmable Logic Array)
      - 2.6.1.2. PLA (Programmable Logic Array)
      - 2.6.1.3. GAL (Generic Array Logic)
    - 2.6.2. CPLD (Complex Programmable Logic Device)
    - 2.6.3. FPGA (Field Programmable Gate Array)
      - 2.6.3.1. Memoria de configuración
  - 2.7.**Fabricantes de dispositivos lógica programable
  - 2.8.**Metodologías y herramientas de diseño de lógica programable
  - 2.9.**JTAG
  - 2.10.** Estado actual de las FPGAs
  3. Detalles del proceso
    - 3.1.**Etapa 1: diseño de la ALU en Logisim (simulador)
      - 3.1.1. Operaciones aritméticas de 4 bits
        - 3.1.1.1. Sumador de 4 bits (4-Bit Full Adder)

- 3.1.1.2. Restador de 4 bits (4-Bit Subtractor)
- 3.1.1.3. Incrementador de 4 bits (4-Bit Incrementer)
- 3.1.1.4. Decrementador de 4 bits (4-Bit Decrementer)
- 3.1.2.** Operaciones lógicas de 4 bits
  - 3.1.2.1. AND de 4 bits (4-Bit AND)
  - 3.1.2.2. OR de 4 bits (4-Bit OR)
  - 3.1.2.3. XOR de 4 bits (4-Bit OR)
  - 3.1.2.4. Complemento a uno de 4 bits (4-Bit One's Complement)
- 3.1.3.** Operaciones de desplazamiento de 4 bits
  - 3.1.3.1. Desplazamiento lógico a la izquierda de 4 bits (4-Bit Left Shift)
  - 3.1.3.2. Desplazamiento lógico a la derecha de 4 bits (4-Bit Right Shift)
  - 3.1.3.3. Rotación a la izquierda de 4 bits (4-Bit Left Rotate)
  - 3.1.3.4. Rotación a la derecha de 4 bits (4-Bit Right Rotate)
- 3.1.4.** Circuitos útiles
  - 3.1.4.1. Multiplexor de 4 a 2 (4-To-2 Multiplexer)
  - 3.1.4.2. Multiplexor de 16 a 4 (16-To-4 Multiplexer)
- 3.1.5.** Resultado final: ALU de 4 bits (4-Bit ALU)

**3.2.** Etapa 2: transferencia del diseño en Logisim a IceStudio para su implementación en la FPGA

3.2.1. Pila de herramientas libres

- 3.2.1.1. IceStudio (editor)

**3.3.** Etapa 3: montaje del sistema y pruebas en la FPGA

- 3.3.1. IceZUM Alhambra
- 3.3.2. Impresión de piezas 3D
- 3.3.3. PCB (*Printed Circuit Board*)

4. Conclusiones y trabajos futuros

5. Referencias

6. Anexo

**6.1.** Ejemplo de la datasheet de una ALU real de, en este caso, Signetics

**6.2.** Esquemas de circuitos

# 1. Introducción

## 1.1. Motivación

El mundo está cambiando y con él las formas en las que las personas conectan y se comunican entre sí. Todo lo que nos rodea y se puede cuantificar se digitaliza de alguna manera, sea los sitios por los que transitamos al día, los pasos que damos si salimos a dar un paseo, el tráfico que nos encontramos en una carretera, etc. Y todo son datos. Datos que, por lo general, van a parar a un centro de procesamiento de datos, se analizan y los resultados de ese procesamiento son devueltos al usuario. Así, no es casualidad que empresas punteras en su campo como Intel hayan decidido introducirse en el mundo del hardware reconfigurable, que es el que nos interesa en nuestro contexto. Para 2020 se estima que, de media, una persona podrá generar 1.5 Gb de información al día. Un coche autónomo generará 4.000 Gb y una fábrica podrá generar hasta 1.000.000 Gb de información. La memoria donde se almacenará esa información es un aspecto crítico. Pero, ¿cómo se va a manejar el procesamiento de tal cantidad de datos? Esta es la razón por la que Intel quiere aprovechar la aceleración que pueden proporcionar las FPGAs al proceso de procesamiento de datos. Gracias al alto nivel de integración hardware de estos dispositivos y a su capacidad de reconfiguración y, por lo tanto, ahorro en costes, el nivel de paralelismo que aportan es masivo y de extraordinaria utilidad. Que empresas como Intel apuesten por esta tecnología, al igual que algunas otras grandes del panorama actual, nos hace pensar que puede tener un gran futuro.

## 1.2. Objetivo

En un deseo por parte de la autora del presente documento de investigar y conocer más a fondo la relación entre software y hardware, y aprovechando la popularidad creciente de las FPGAs, que dada su naturaleza permite trabajar con hardware y aspectos de bajo nivel, se va a realizar el diseño del circuito digital de una Unidad Aritmético Lógica (*ALU, Arithmetic Logic Unit*) y se va a implementar en una FPGA. Para llevar a cabo este trabajo, se han necesitado repasar conceptos de

electrónica digital, partiendo de los componentes básicos hasta llegar a cómo combinarlos para formar circuitos complejos. Todo esto usando herramientas software libres.

### 1.3. Visión general del proceso de desarrollo del proyecto

Valorando diferentes ideas por las que comenzar a investigar, se decidió profundizar en el mundo de las FPGAs. Un vídeo de Internet, de Juan González, doctor en Robótica por la Universidad Autónoma de Madrid y actualmente profesor e investigador en la Universidad Rey Juan Carlos de Madrid, en la que explicaba de forma muy sencilla y amena qué eran las FPGAs, para qué servían y cómo se usaban reavivaron el interés de la autora en el tema [1]. Esta fuente sirvió como punto de partida para adentrarme en la comunidad y descubrir que estaban trabajando en una tarjeta de desarrollo con una FPGA integrada llamada IceZUM Alhambra. El movimiento y actividad que se apreciaba fue un incentivo para conocer más a fondo esta tecnología.

Originalmente se planteó diseñar un filtro digital, pero requería unos conocimientos más avanzados. Así que se optó por hacer algo más relacionado con el grado cursado y la informática, en este caso una Unidad Aritmetico Lógica o ALU. Hacer una ALU sencilla puede que no sea una idea muy original, pero lo más interesante radicaba en conocer en qué consistía una FPGA a nivel de hardware, cómo implementar circuitos digitales en ella y, a través de la introducción de bits por los pines de entrada de la placa crear, aunque sencillo, nuestro propio lenguaje de programación a muy bajo nivel.

Durante el desarrollo del proyecto, como autora del mismo siento la necesidad de mencionar un libro que me ha sido de gran utilidad. Se titula “*Code: The Hidden Language of Computer Hardware and Software*” (Charles Petzold, 2000, Microsoft Press), en el cual, Petzold narra de forma ingeniosa y familiar los entresijos de los ordenadores y otras máquinas. Relatando la historia de la computación, resaltando la importancia de la existencia de un código de comunicación, describiendo el funcionamiento los relés, pasando por los transistores hasta construir puertas lógicas, biestables, memoria, procesadores y circuitos integrados, así como la invención del lenguaje ensamblador y de los lenguajes de alto nivel, hasta el funcionamiento de los gráficos de las pantallas [8]. Además de esta lectura a nivel teórico, se investigó más a fondo sobre dispositivos lógicos programables (PLDs), de los cuales un tipo es la FPGA.

También se han imprimido piezas 3D, descargando el diseño desde la plataforma de la comunidad para construir interruptores y se han elaborado placas de circuitos impresos (*PCBs, printed circuit boards*) que servirán de entrada de datos a la FPGA y a la ALU implementada en ella.

# 2. Estado del arte

## 2.1. Circuitos digitales

Si observamos el interior de los chips digitales, veremos hay muchos niveles. Nosotros nos vamos a quedar en el primer nivel que es el de electrónica digital. Por debajo encontramos los transistores, el silicio, los átomos, etc. Pero a nivel de electrónica digital, los chips son algo muy sencillo: son unos elementos que trabajan con números binarios, 0 y 1, y lo único que hacen es manipular, almacenar y transportar estos bits. No hacen nada más.

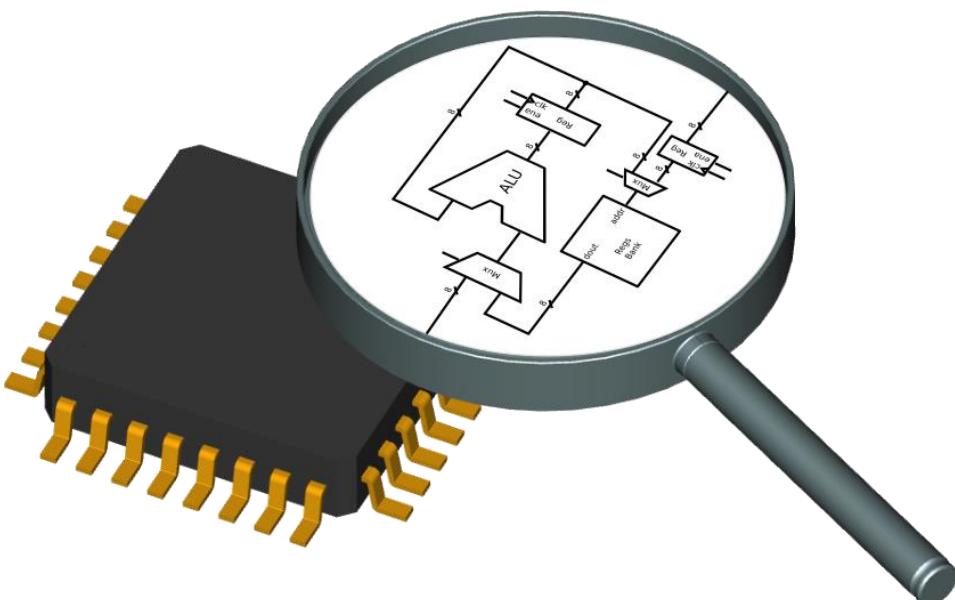
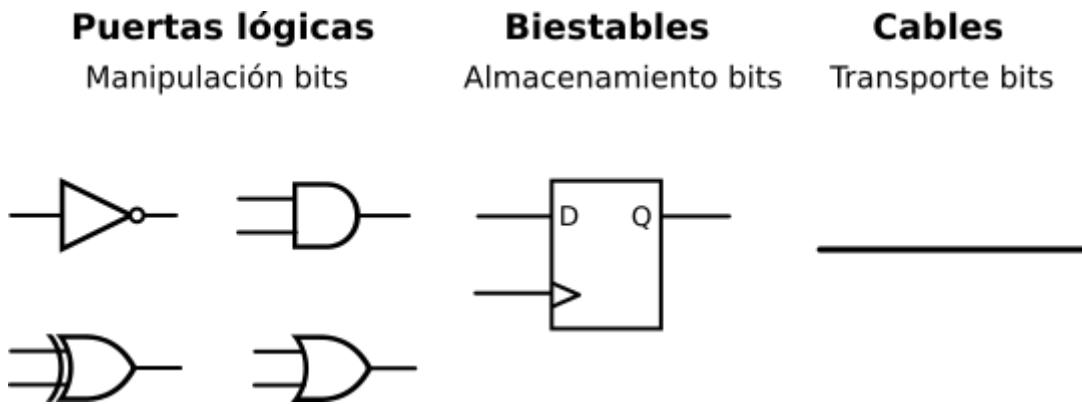


Figura 1. Representación simbólica del interior de un chip. Nota. Recuperado de “Introducción”, de FPGA Wars. Recuperado de <http://obijuan.github.io/intro-fpga.html>

Y no importa lo complejo que sea el chip, podemos coger el último Intel, ARM o AMD. Todos al final se basan en lo mismo. La electrónica digital se basa **en tres componentes elementales**: **puertas lógicas**, que nos permiten **manipular** los bits; **biestables**, que nos permiten **almacenar** los bits; y **cables**, que nos permiten unir los componentes y **transportar** los bits. **No hay nada más. Todo se construye a partir de estos elementos tan sencillos.** Si unimos biestables tenemos un registro, si unimos varios registros obtenemos una memoria, con puertas lógicas podemos hacer multiplexores, unidades aritmético-lógicas (ALU) (lo que vamos a hacer nosotros), etc. Todo va creciendo, basándose en estos tres elementos. Entonces, combinándolos inteligentemente, podemos crear nuestros circuitos digitales. [2]



*Figura 2. Elementos básicos de los circuitos digitales. Nota. Recuperado de “Introducción”, de FPGA Wars. Recuperado de <http://obijuan.github.io/intro-fpga.html>*

En su forma más simple, la **lógica** es la parte del razonamiento humano que nos dice que una determinada proposición (sentencia de asignación) es cierta si se cumplen ciertas condiciones. Muchas situaciones del día a día pueden expresarse como funciones proposicionales o lógicas. Dado que tales funciones son sentencias verdaderas/falsas o afirmativas/negativas, pueden aplicarse a los circuitos digitales, ya que estos se caracterizan por sus dos estados. El término “lógico” se aplica a circuitos digitales que se utilizan para implementar funciones lógicas. Hacia 1850, el matemático y lógico irlandés George Boole desarrolló un sistema matemático para formular proposiciones lógicas con símbolos, de manera que los problemas pudieran formularse y resolverse de forma similar a como se hace en el álgebra ordinaria. Tal rama se conoce hoy como álgebra de Boole [23].

**Los primeros circuitos electrónicos eran analógicos**, y el procesamiento de señales era llevado a cabo por ellos. La invención del transistor semiconductor en 1947 en los Laboratorios Bell, la mejora de estos, la llegada de los circuitos integrados o CIs (*integrated circuits, ICs*) lineales (analógicos), los primeros CIs digitales basados en TTL (*transistor-transistor logic*) en los años 60, seguidos pronto por los CIs con tecnología CMOS (*complementary metal-oxide-semiconductor*). Los primeros dispositivos incorporaban un número pequeño de puertas lógicas (nivel de densidad bajo), pero el crecimiento de este número llevó al microprocesador en los años

70. Además, la habilidad para crear CIs con características de memoria llevó a la expansión en la industria de los ordenadores y a los tipos de sistemas digitales complejos basados en la arquitectura de los ordenadores que tenemos disponibles hoy en día. Los últimos sesenta años han visto una revolución en la industria de la electrónica.

Básicamente, un circuito digital se puede clasificar en tres tipos generales:

- **Lógica combinacional**, en la que la respuesta del circuito está basada solo en una expresión lógica booleana de la entrada y el circuito responde inmediatamente a un cambio en la entrada.
- **Lógica secuencial**, en la que la respuesta del circuito está basada en el estado actual del circuito y algunas veces en la entrada actual. A su vez, puede ser **síncrono** o **asíncrono**, dependiendo de si los cambios en el estado se producen a causa de una señal de reloj o si el circuito no utiliza un reloj, respectivamente.
- **Memoria**, en la que los valores digitales pueden ser almacenados y accedidos algún tiempo después. Puede ser de *solo lectura* (*read-only memory, ROM*) o de *acceso aleatorio* (*random access memory, RAM*). [7]

## 2.2. Lógica programable

En lógica fija, las funciones lógicas son definidas por el fabricante y no se pueden modificar. Por ejemplo, el microprocesador que incluye cualquier ordenador convencional se basa en lógica fija porque no podemos cambiar su arquitectura ni funcionamiento interno por nuestra cuenta. Cuando el hardware es de tipo específico, se denomina **ASIC** (*Application Specific Integrated Circuit*).

**El problema hasta ahora era la implementación.** Se diseñaba un circuito en papel u ordenador, se simulaba, pero en el momento de crearlo **físicamente** con piezas reales, dependiendo de su complejidad, se podía convertir en una tarea ardua. **Por esto surgió la lógica programable.**

Es una idea muy sencilla: vamos a coger los tres componentes básicos de los circuitos digitales, esto es, las puertas lógicas, los biestables y los cables, y los vamos a meter en un chip, sin conectar. Entonces, inicialmente están “**en blanco**”, entonces decimos que el dispositivo no está “**configurado**”, porque sus elementos internos se encuentran desconectados. Pero, de repente podemos decir “queremos hacer estas uniones”, como resultado nos aparece el circuito. Es decir, **tenemos circuitos digitales bajo demanda**. Ya sea un controlador de VGA, un microprocesador o un ALU lo que queramos implementar, lo podemos hacer, solo tenemos que especificar la conexión entre los diferentes elementos.

El hardware usado para implementar circuitos de esta forma se conoce como **dispositivo lógico programables** (*Programmable Logic Device, PLD*).

Ahora que tenemos el concepto básico de lógica programable, podemos hacer una aclaración: aunque los ASICs pueden entenderse como dispositivos de lógica fija porque vienen configurados de fábrica, también es cierto que existen ASICs que, en muchos casos, no son dispositivos completamente configurados completamente de fábrica (en cuyo caso se adjetivarían como *full-*

*custom*), sino que son parcialmente modificables (*semi-custom*) lo que aumenta el número de aplicaciones en que podrían ser usados y justifica su clasificación como dispositivo lógico programable. [15]

También es frecuente encontrarnos con sistemas en un chip (*System on a Chip, SoC*), que combinan procesador, distintos tipos de memorias e incluso hardware programable como una FPGA en un único dispositivo. Conseguimos varias mejoras: una mayor integración, un tamaño físico menor y un ancho de banda mayor entre el procesador y la FPGA (ver Figura 2). Un ejemplo de un SoC es el *Stratix 10* de Intel en colaboración con Altera (año 2016).

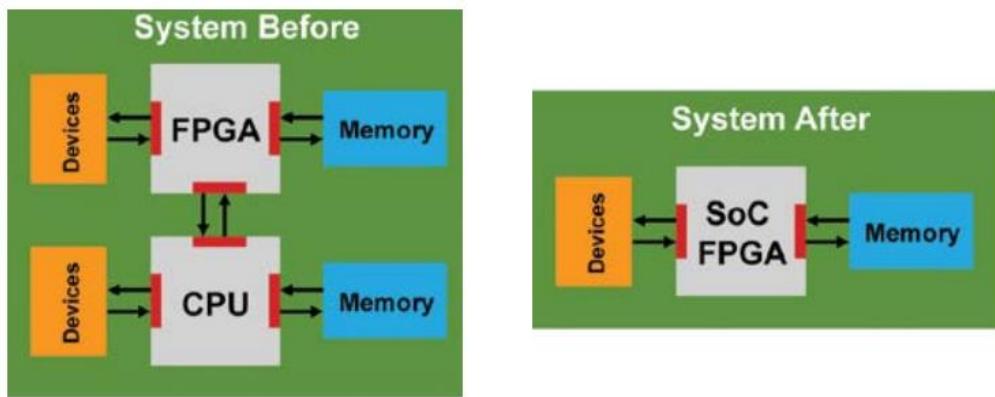


Figura 3. Comparación entre una conexión entre un procesador *standalone*, celdas de memoria y una FPGA, y un SoC con el procesador y la FPGA emplazados en un único chip, con las respectivas ventajas de esta arquitectura.

Nota. Recuperado de “*What is an SoC FPGA?*”, de Altera, 2014. Recuperado de  
[https://www.altera.com/ja\\_JP/pdfs/literature/ab/ab1\\_soc\\_fpga.pdf](https://www.altera.com/ja_JP/pdfs/literature/ab/ab1_soc_fpga.pdf)

En las siguientes figuras 2 y 3 podemos ver nuestro PLD configurado de dos formas distintas:

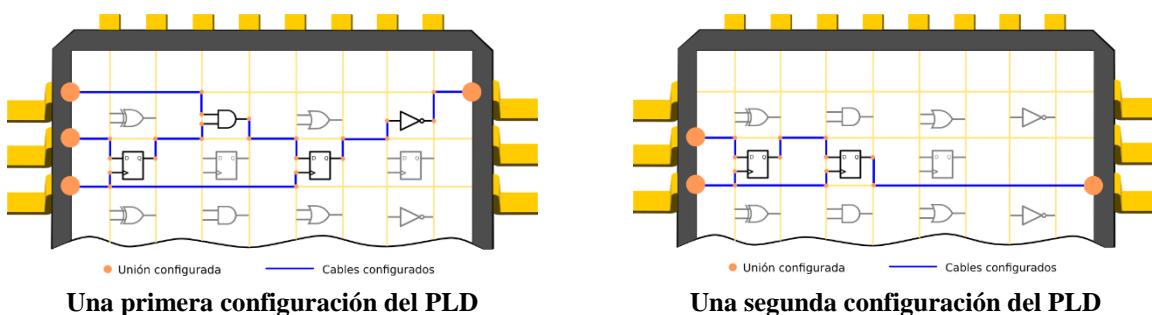


Figura 4. Dos configuraciones diferentes del PLD. Nota. Recuperado de “*Introducción*”, de FPGA Wars.  
 Recuperado de <http://objijuan.github.io/intro-fpga.html>

Igual que en software podemos cambiar unas líneas de código por otras y obtenemos un programa completamente diferente, podemos cambiar las conexiones que existen dentro de un circuito para tener un componente totalmente diferente. Para ello, como veremos, haremos uso de un software especial conectado a nuestro dispositivo programable.

Sin embargo, la lógica programable permite que el fabricante (por ejemplo, para prototipar antes de pasar a producción) o el usuario en su casa puedan implementar sus diseños lógicos y modificarlos fácilmente sin tener que volver a cablear o reemplazar componentes.

Como desventaja, lo que podemos implementar en un PLD está limitado por el espacio, realmente por el tamaño de la matriz de componentes (no podemos hacer un circuito con treinta puertas lógicas si nuestra matriz solo dispone de veinte). Pero al final tenemos un chip con el circuito que queremos.

Esos sí, si el circuito lo fabricásemos como un chip específico (lo que se conoce como *ASIC*, *Application Specific Integrated Circuit*) para nuestra aplicación, va a ser óptimo, más rápido, más eficiente, etc. La FPGA puede que no nos dé estas características en ciertas aplicaciones como lo haría el ASIC, pero es una **aproximación** muy buena, por lo que también es muy útil para **prototipar**.

En resumen, a la hora de diseñar un circuito lógico, una de las decisiones que deberemos tomar será si usar dispositivos de lógica fija (aquellos que tendrán una funcionalidad no modificable) o usar un dispositivo lógico programable (PLD). La elección dependerá de los requisitos del diseño en concreto. Por ejemplo, si necesitamos un circuito con solo unas cuantas puertas lógicas, una implementación de lógica fija nos puede bastar. Sin embargo, si quisieramos un circuito digital más complejo como el diseño de un filtro digital, entonces, teniendo en cuenta la complejidad del hardware, un PLD sería una opción para tener muy en cuenta.

### 2.3. ALU (*Arithmetic Logic Unit*)

El procesador o CPU está formado por varios componentes. Uno es la ALU, un circuito digital de lógica fija. Se encarga de realizar **operaciones aritméticas** (suma, resta, multiplicación, división, etc.) y **operaciones lógicas** (AND, OR, XOR, etc.) sobre distintos valores. La ALU podríamos decir que es el cerebro matemático de un ordenador. Representa el bloque principal dentro de la **unidad central de procesamiento (CPU)** del ordenador. Además de la ALU, las CPUs modernas tienen una **unidad de control (CU)**.

La ALU carga datos desde registros. Un registro es una pequeña cantidad de almacenamiento disponible como parte de la CPU. La unidad de control ordena a la ALU qué operación debe realizar y cuándo sobre un dato. Entonces la ALU guarda el resultado en un registro de salida. La unidad de control mueve los datos entre estos registros, la ALU y la memoria.

### 2.3.1. Historia

J. Presper Eckert (1919-1995) y John Mauchly (1907-1980) de la Universidad de Pensilvania diseñaron el **ENIAC** (*Electronic Numerical Integrator and Computer*). Usaba 18.000 tubos de vacío y se completó en 1945.

El ENIAC llamó la atención del matemático John von Neumann (1903-1957). Con reputación por hacer complejos cálculos aritméticos mentales, ejercía de profesor en el Princeton Institute for Advanced Study, e investigó en todo desde Mecánica Cuántica hasta la aplicación de la Teoría de Juegos en economía.

John von Neumann ayudó en el diseño del **EDVAC** (*Electronic Discrete Variable Automatic Computer*), el sucesor del ENIAC. En un paper publicado en 1946 titulado “Preliminary Discussion of the Logical Design of an Electronic Computing”, coautorizado con Arthur W. Burks y Herman H. Goldstine, describió varias características de un ordenador que supuso un avance respecto al ENIAC. Los diseñadores del EDVAC creían que el ordenador debía usar **números binarios** internamente. El ENIAC usaba números decimales. Además, debía tener la mayor cantidad de memoria posible para almacenar el código del programa y los datos mientras el programa se ejecutaba (el ENIAC no funcionaba así, programarlo era cuestión de mover interruptores y cables de un lado a otro). Estas instrucciones debían ser secuenciales en memoria y direccionadas a través de un contador, pero también debía permitir saltos condicionales.

Estas decisiones de diseño fueron un paso tan importante que hoy día hablamos de “**arquitectura de von Neumann**”. Aun así, existía un problema conocido como el “**cuello de botella de von Neumann**”. Y es que una máquina con esta arquitectura generalmente invierte una cantidad de tiempo importante en hacer un *fetch* de las instrucciones desde la memoria en el proceso de preparación para ejecutarlas. [8] Una mitigación frente a esto podría ser utilizar una memoria caché entre la CPU y la memoria principal. [13]



Figura 5. John von Neumann (1903-1957). Nota. Recuperado de “John von Neumann”, Wikipedia. Recuperado de [https://es.wikipedia.org/wiki/John\\_von\\_Neumann](https://es.wikipedia.org/wiki/John_von_Neumann) [12]

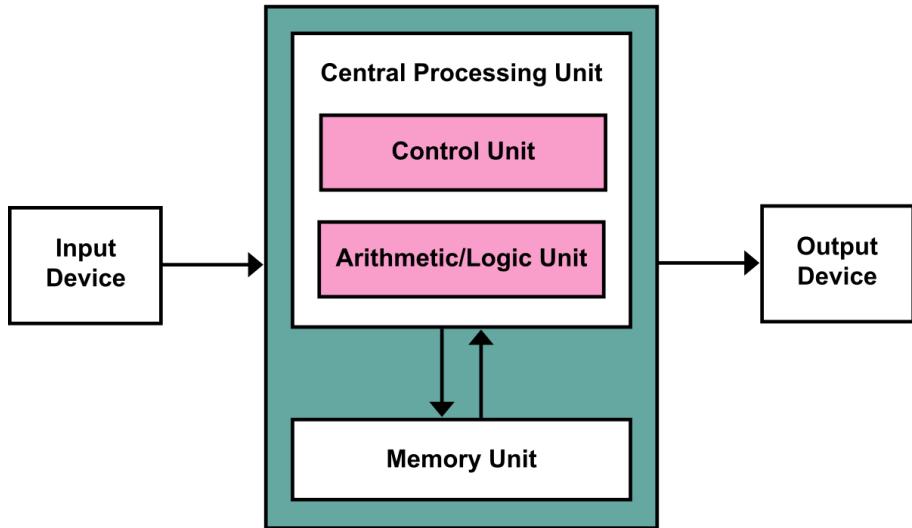


Figura 6. Diagrama simplificado de la arquitectura de von Neumann. Nota. Recuperado de “Von Neumann architecture”, Wikipedia. Recuperado de [https://en.wikipedia.org/wiki/Von\\_Neumann\\_architecture](https://en.wikipedia.org/wiki/Von_Neumann_architecture)

### 2.3.2. Funcionamiento general

Una ALU es un circuito combinacional, significando que las salidas cambian asíncronamente con las entradas. Aplicamos unas señales a las entradas, esperamos un tiempo (conocido como “**tiempo de propagación**”) a que esas señales “cruzen” la circuitería de la ALU y el resultado aparece en la salida.

Para asegurar que no entran señales a un ritmo más rápido que el tiempo de propagación (lo cual podría producir resultados inesperados en la salida), utilizamos una **señal de reloj**. [10]

En la Figura 6 vemos la circuitería de la primera ALU implementada en un único chip, esto es, un circuito integrado. Opera con 4 bits. Varios fabricantes produjeron este chip, tales como Motorola, Fairchild o Signetics. En el anexo se añade un enlace a la hoja de especificaciones del chip del último, a la cual pertenece la circuitería mostrada [14]:

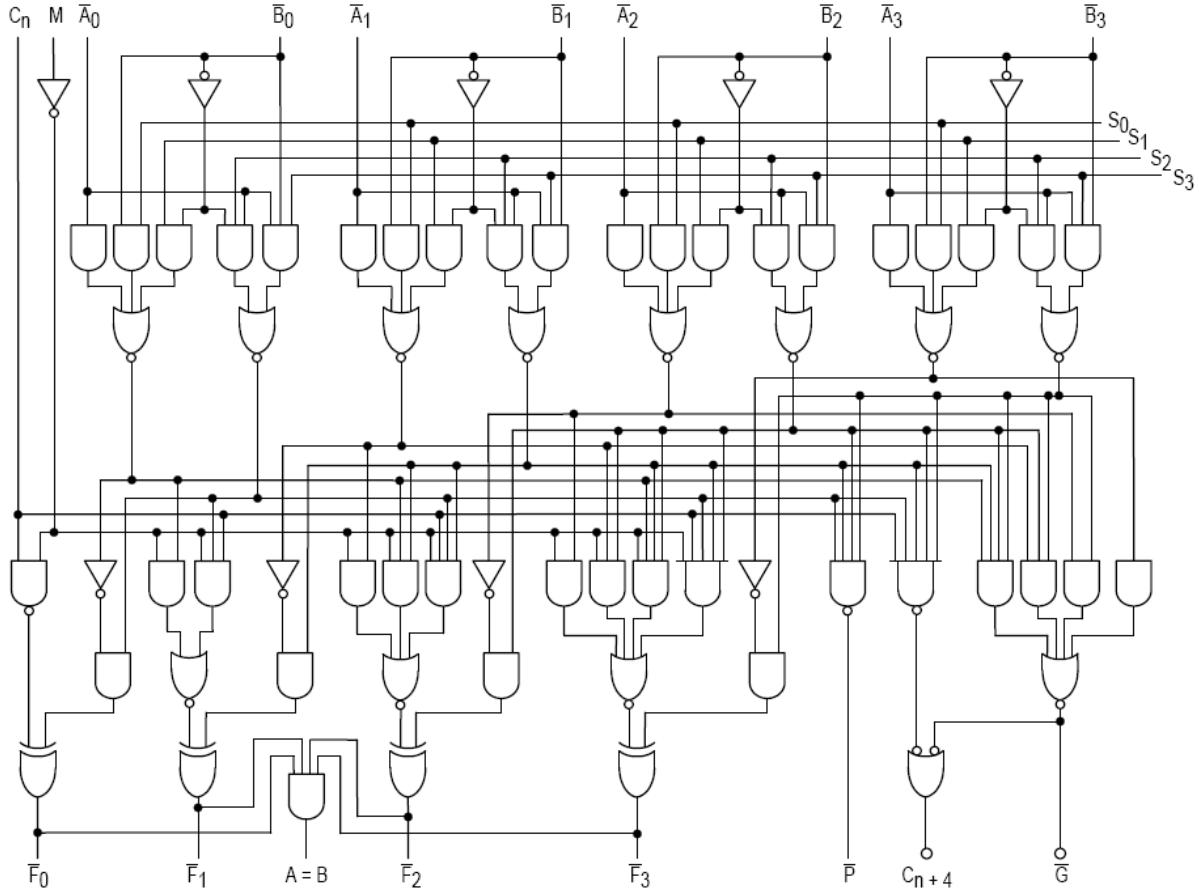


Figura 7. Circuitería del IC 74181 de Signetics, una ALU de 4 bits. Nota. Recuperado de “74181”, Wikipedia. [11] Recuperado de <https://en.wikipedia.org/wiki/74181>. En el Anexo se añade un enlace a la hoja de especificaciones o *datasheet* del este IC de Signetics

### 2.3.3. Operaciones

#### 2.3.3.1. Operaciones aritméticas

Las operaciones aritméticas típicamente implementadas en una ALU son: suma, suma con acarreo, resta, resta con llevado, incremento, decremento, complemento a dos. [10]

Para simplificar, en este trabajo implementaremos suma, resta, decremento e incremento.

#### 2.3.3.2. Operaciones lógicas

A nivel lógico nos encontramos: AND, OR, XOR, complemento a uno. [10]

En este trabajo, implementaremos estas cuatro funciones.

### 2.3.3.3. Operaciones de desplazamiento

Nos encontramos con distintos tipos de desplazamientos: aritmético, lógico, rotación, todas tanto a izquierda como a derecha. [10]

Para simplificar, en este trabajo implementaremos desplazamiento y rotación, ambos a izquierda y a derecha.

## 2.4. Concepto básico de la matriz AND

La mayor parte de los PLDs usan alguna forma de **matriz AND**. Esta matriz está formada, básicamente, por puertas AND y una matriz o red de interconexiones con conexiones programables en cada punto de intersección, como se muestra en la primera figura. El propósito de las conexiones programables (que nosotros podemos quitar o poner, como queramos) es establecer o interrumpir una conexión entre una fila y una columna de la matriz de interconexión, como podemos ver en la segunda figura. Las columnas suelen ser las señales de entrada (en el ejemplo de la figura 7 serían: A, A negada, B y B negada). En las figuras 7 podemos ver una matriz AND genérica no programada y programada, respectivamente:

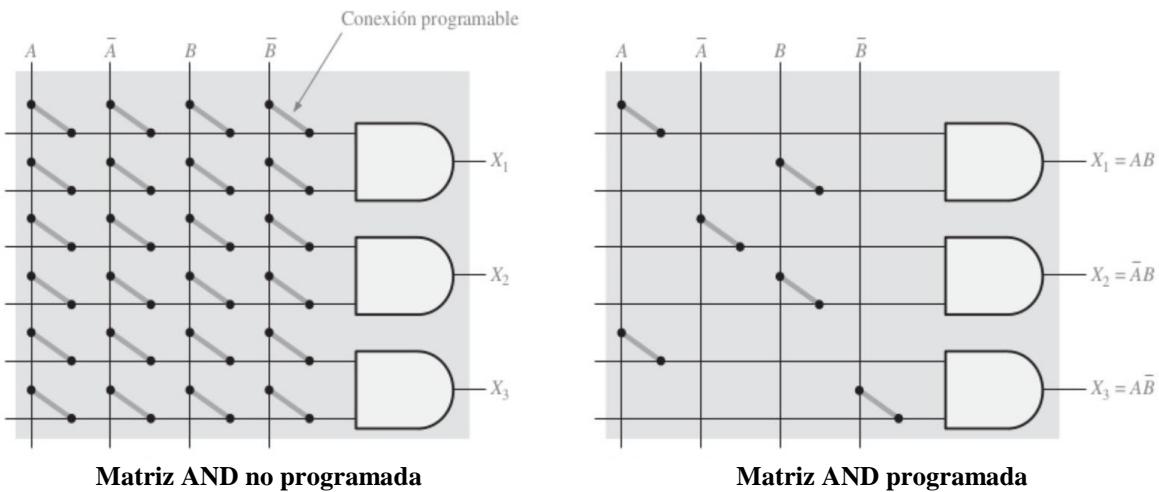


Figura 8. Matriz AND genérica. Nota. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 156, Madrid, España: Pearson Educación S.A.

## 2.5. Tecnologías de proceso basadas en conexiones programables

Hay varias tecnologías para programar una matriz como la que acabamos de ver.

### 2.5.1. Tecnología basada en fusible

Fue la tecnología **original** usada en los SPLDs. Consiste en un **fusible** que, al principio, conecta una columna con una fila. A partir de aquí, nosotros podemos considerar “fundir” el fusible haciendo pasar una corriente determinada y romper la conexión o dejar el fusible intacto y mantener la conexión.

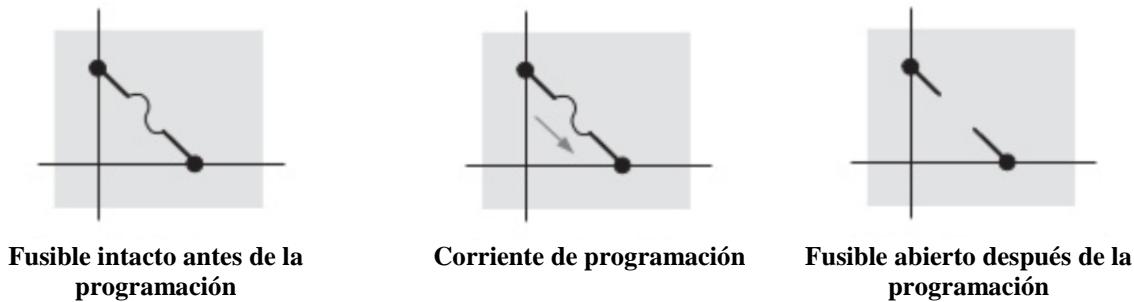


Figura 9. Conexión programable mediante fusible. Nota. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 157, Madrid, España: Pearson Educación S.A. [9]

### 2.5.2. Tecnología basada en antifusible

Es lo opuesto a una conexión mediante fusible. Al principio, el **antifusible** es un **circuito abierto** (por lo tanto, no deja pasar corriente) entre una columna y una fila. Un antifusible consiste en dos conductores separados por un aislante. A partir de aquí, nosotros podemos considerar romper el aislamiento entre los conductores, de modo que, aplicando una tensión adecuada al antifusible, el aislante pase a ser una conexión de baja resistencia o mantener este último intacto y mantener la conexión.

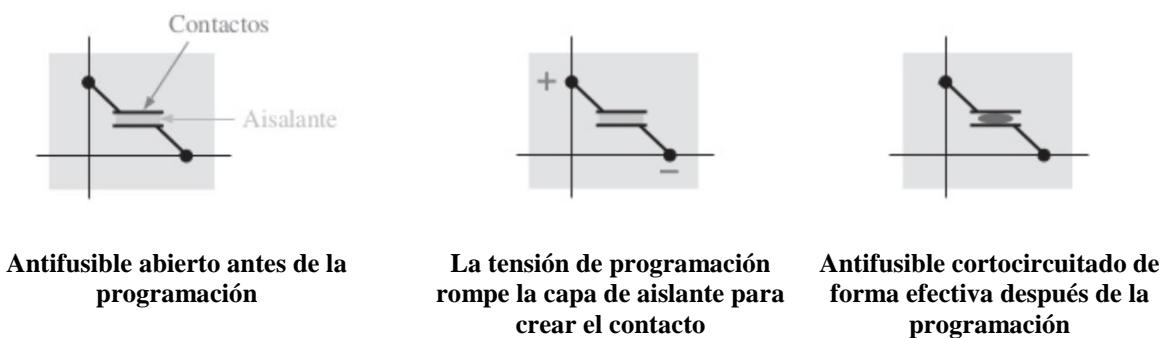


Figura 10. Conexión programable mediante antifusible. Nota. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 157, Madrid, España: Pearson Educación S.A.

### 2.5.3. Tecnología basada en EPROM (*Electrically Programmable Read-Only Memory*)

En este caso, las conexiones programables son similares a las celdas de memoria de las EPROM. Emplea un tipo especial de **puerta MOS**, conocido como **transistor de puerta flotante**, como conexión programable. Usa un proceso denominado **Fowler-Nordheim** para colocar electrones en la estructura de puerta flotante. La mayoría de los PLDs que usan esta tecnología son programables una sola vez (*one-time programmable, OTP*), a no ser que dispongan de un **encapsulado de ventana**. En este caso, la configuración se puede borrar utilizando **luz ultravioleta (UV)** y volverse a programar de forma normal.

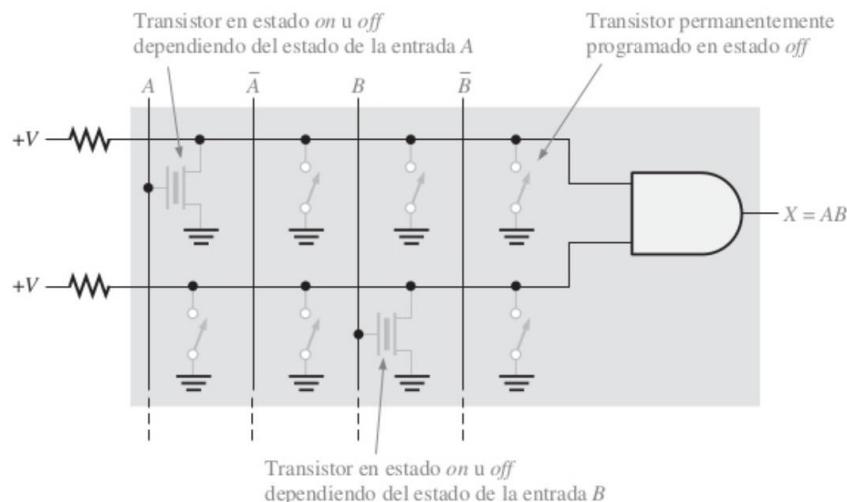


Figura 11. Tecnología basada en EPROM. Nota. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 158, Madrid, España: Pearson Educación S.A.

### 2.5.4. Tecnología basada en EEPROM (*Electrically Erasable Programmable Read-Only Memory*)

Como la tecnología EPROM con la diferencia de que no nos hace falta luz UV para borrar la configuración. **Podemos programar y reprogramar usando electricidad.**

### 2.5.5. Tecnología basada en SRAM (*Static Random Access Memory*)

Emplea una **celda de memoria tipo SRAM** para activar o desactivar un transistor con el fin de conectar o desconectar las filas y columnas. Por ejemplo, cuando la celda de memoria contiene un 1 (en gris), el transistor se **activa (on)** y conecta la fila y columna asociadas, como se muestra en la imagen a la izquierda de la figura 11. Si la celda de memoria contiene un 0 (en negro), el

transistor se **desactiva (off)** y no se establece ninguna conexión entre la fila y columna asociadas, como se muestra en la imagen a la derecha de la misma figura.

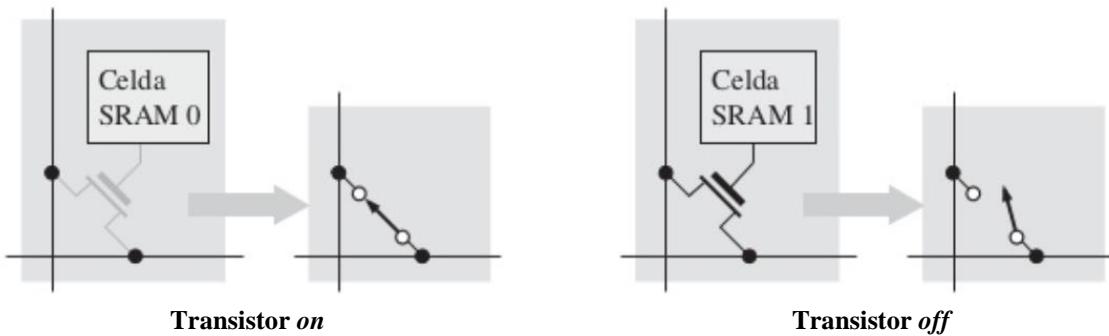


Figura 12. Tecnología basada en SRAM. Nota. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 159, Madrid, España: Pearson Educación S.A.

## 2.6. Tipos de dispositivos lógicos programables (Programmable Logic Devices, PLDs)

Existen muchos tipos de dispositivos lógicos programables, desde pequeños dispositivos que pueden reemplazar a algunos de los dispositivos de función fija hasta complejos dispositivos de alta densidad o nivel de integración que pueden reemplazar a miles de dispositivos de función fija. A continuación, veremos los principales:

### 2.6.1. SPLD (Simple Programmable Logic Device)

El SPLD (del inglés, *Simple PLD*) se corresponde con el PLD original y, generalmente, puede reemplazar a un número pequeño de CIIs de función fija. Son PLDs sencillos, en el sentido de disponer de un número relativamente pequeño de puertas lógicas. Los tres tipos principales de arquitectura SPLD son: PLA, PAL y GAL:

#### 2.6.1.1. PAL (Programmable Logic Array)

Las matrices lógicas programables fueron los primeros PLDs (en concreto, SPLDs) con una presencia muy relevante en el mercado. Consta de una matriz programable de puertas AND que se conecta a una matriz fija de puertas OR. Permite cualquier expresión lógica de tipo **suma de productos (SOP, sum-of-products)**. Con esto nos ahorraremos el tiempo de propagación relacionado con el plano OR, consiguiendo un diseño más rápido. Sin embargo, perdemos flexibilidad a la hora de implementar el circuito.

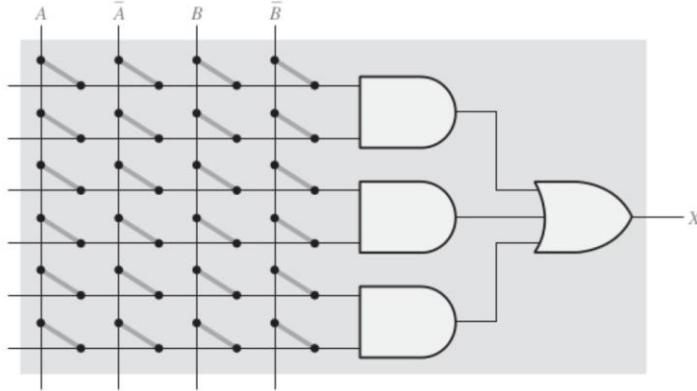


Figura 13. Arquitectura PAL. Nota. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 682, Madrid, España: Pearson Educación S.A.

### 2.6.1.2. PLA (Programmable Logic Array)

El PLA consiste en un plano AND programable y un plano OR también programable.

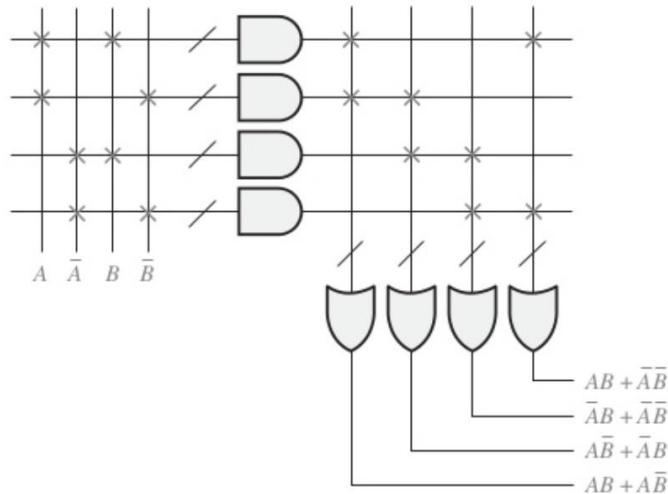


Figura 14. Arquitectura PLA. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 698, Madrid, España: Pearson Educación S.A.

### 2.6.1.3. GAL (Generic Array Logic)

PLA y PAL son dispositivos programables una sola vez (OTP) basados en PROM, por lo que su configuración no puede ser cambiada una vez ha sido establecida. Esto significa que, si queremos cambiar el diseño de nuestro circuito, tendremos que tirar nuestro dispositivo a la basura y utilizar uno nuevo. Aquí es donde entra GAL, similar a PAL pero que usa EEPROM (E<sup>2</sup>PROM) en lugar de fusibles y puede, por tanto, ser reconfigurada.

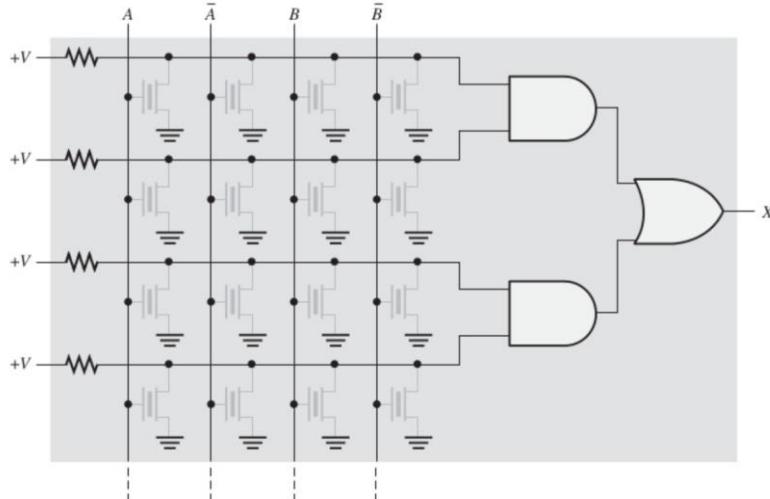


Figura 15. Arquitectura GAL. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 684, Madrid, España: Pearson Educación S.A.

## 2.6.2. CPLD (Complex Programmable Logic Device)

El CPLD es un paso más allá de los SPLDs; desde que los fabricantes fueran capaces de incluir más de un SPLD en un mismo chip, nació el CPLD (*Complex PLD*). Al disponer de un nivel de integración aún mayor respecto a los SPLDs, podemos crear diseños digitales más grandes.

Una de las principales diferencias con los SPLDs es la agrupación de las matrices suma de productos (SOP) en **macroceldas** interconectadas y agrupadas **bloques de matriz lógica (LAB, Logic Array Block)**. Estos bloques están conectados con otros a través de una matriz de conmutación, conocida como *PIA (Programmable Interconnection Array)*. Cada LAB equivale aproximadamente a un SPLD. [23]

## 2.6.3. FPGA (Field Programmable Gate Array)

La FPGA es un paso más allá en complejidad y densidad del CPLD pudiendo crear un diseño aún más complejo. Sin embargo, al contrario que el CPLD, la arquitectura FPGA fue desarrollada utilizando un **concepto básico diferente**. Su arquitectura interna está basada en tres elementos principales:

- Bloques lógicos configurables (*CLBs, Configurable Logic Blocks*)
- Matriz de interconexiones programables
- Bloques de entrada salida

Los bloques lógicos de la FPGA no son tan complejos como los del CPLD, pero, generalmente, contiene muchos más. Cando estos bloques son relativamente simples, se dice que la arquitectura

de la FPGA es de “granularidad fina”. Cuando son más complejos, la arquitectura se denomina de “granularidad gruesa”. Los bloques de entrada salida se encuentran en los bordes exteriores de la estructura. La matriz de interconexiones programable distribuida proporciona la interconexión de los bloques lógicos y los bloques de E/S. [23]

En lugar de calcular los resultados de la lógica combinacional de las sumas de productos como en SPLDs y CPLDs, los bloques lógicos de las FPGAs utilizan tablas de búsqueda (*LUT, look-up tables*). Las LUTs permiten almacenar valores arbitrarios que corresponderán al resultado o salida de una función lógica de determinados valores de entrada. [15].

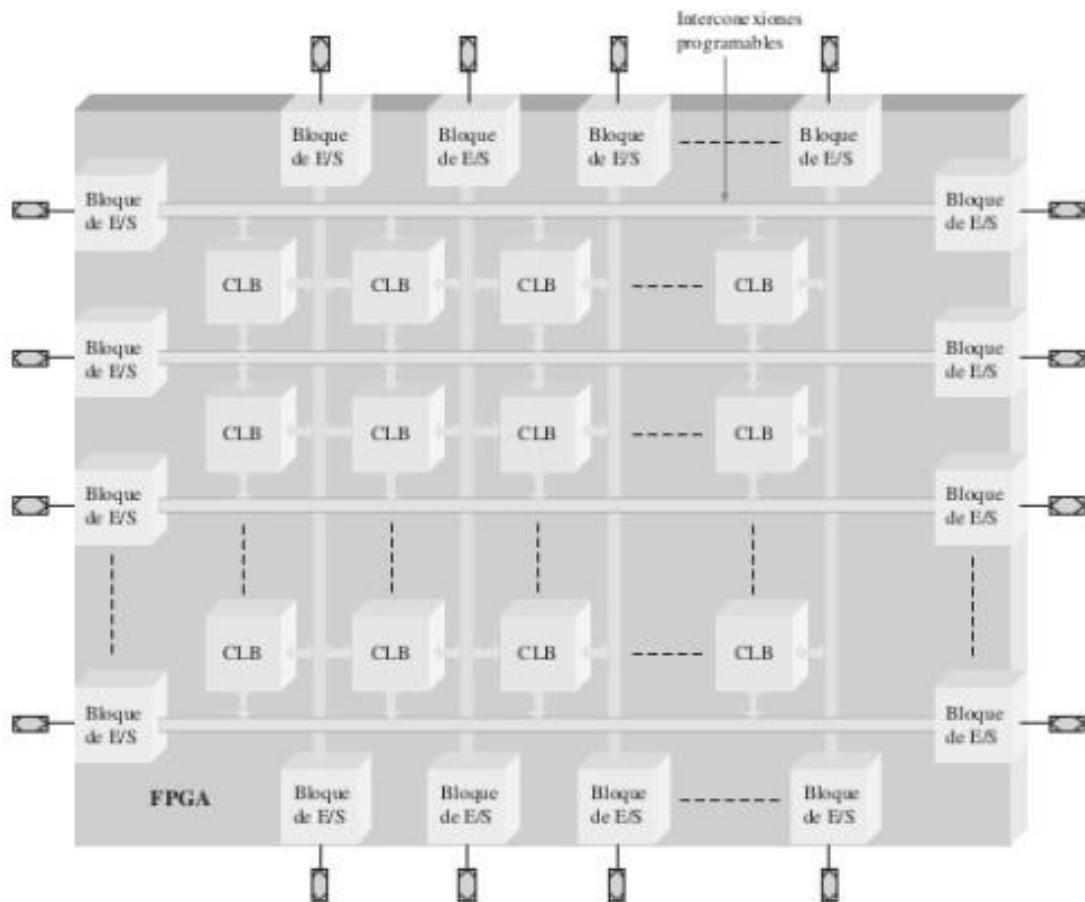


Figura 16. Arquitectura básica de una FPGA. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 707, Madrid, España: Pearson Educación S.A.

Como podemos apreciar en la figura anterior (figura 16), los bloques lógicos o CLBs acceden a los bloques de entrada/salida mediante una matriz de interconexiones programables que los interconectan.

Los CLBs están formados por cierto número de LUTs cuyas salidas, multiplexadas, están conectadas a su vez a biestables y a las salidas del bloque. El nivel de complejidad crece rápidamente al aumentar la capacidad de las LUTs.

### 2.6.3.1. Memoria de configuración

Las FPGAs son reprogramables y utilizan **tecnología de proceso SRAM (volátil) o antifusibles (no volátil)** para implementar las conexiones programables. En el caso de las FPGAs basadas en SRAM, es preciso usar una memoria que o está integrada en la propia FPGA, o bien se utiliza una memoria externa, encargándose un procesador “host” de controlar la transferencia de datos, que puede ser de un PC, una Raspberry Pi, un Arduino, etc. Esto se ilustra en la figura 17.

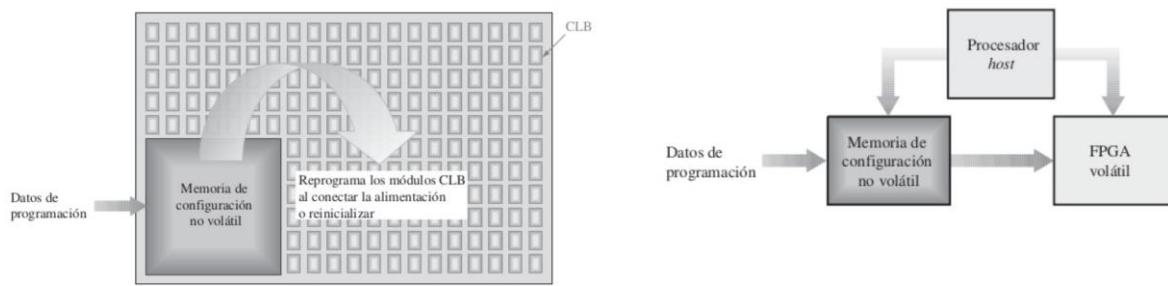


Figura 17. Una memoria de configuración interna al chip (izquierda) y una memoria externa al chip incrustada en la placa que soporta la FPGA. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 711, Madrid, España: Pearson Educación S.A.

Cuando adquirimos una FPGA es posible que traigan lógica implementada en hardware mediante lo que se denomina módulo hardware (ver figura 18). Un **módulo hardware** es una parte de la lógica dentro de una FPGA que el fabricante incluye para proporcionar una función específica y que no puede reprogramarse. Si la función integrada presenta algunas características programables se la conoce con el nombre de **módulo software**. Una ventaja de los módulos hardware es que ahorrán tiempo de desarrollo al usuario. Incluso puede darse una mezcla de módulos hardware y software. Por ejemplo, un procesador que disponga de cierta flexibilidad a la hora de seleccionar y ajustar parámetros por parte del usuario.

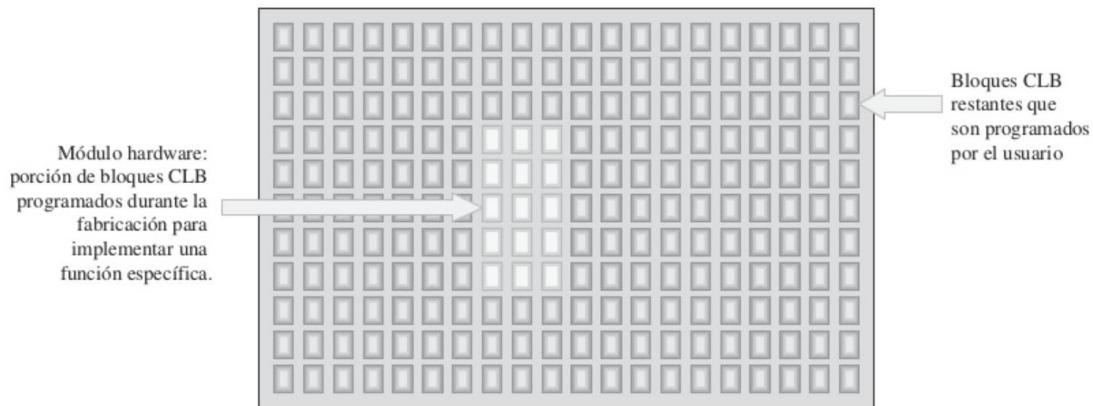


Figura 18. Un módulo hardware incluido en una FPGA. Recuperado de “Fundamentos de sistemas digitales”, de Floyd, Thomas L., 2006, p. 711, Madrid, España: Pearson Educación S.A.

Los diseños de módulos hardware suelen desarrollarlos los fabricantes de FPGAs, son propiedad intelectual de dichos fabricantes. Se denominan **propiedad intelectual (IP, Intellectual Property)**. [23]

El siguiente esquema es un resumen de lo que acabamos de ver:

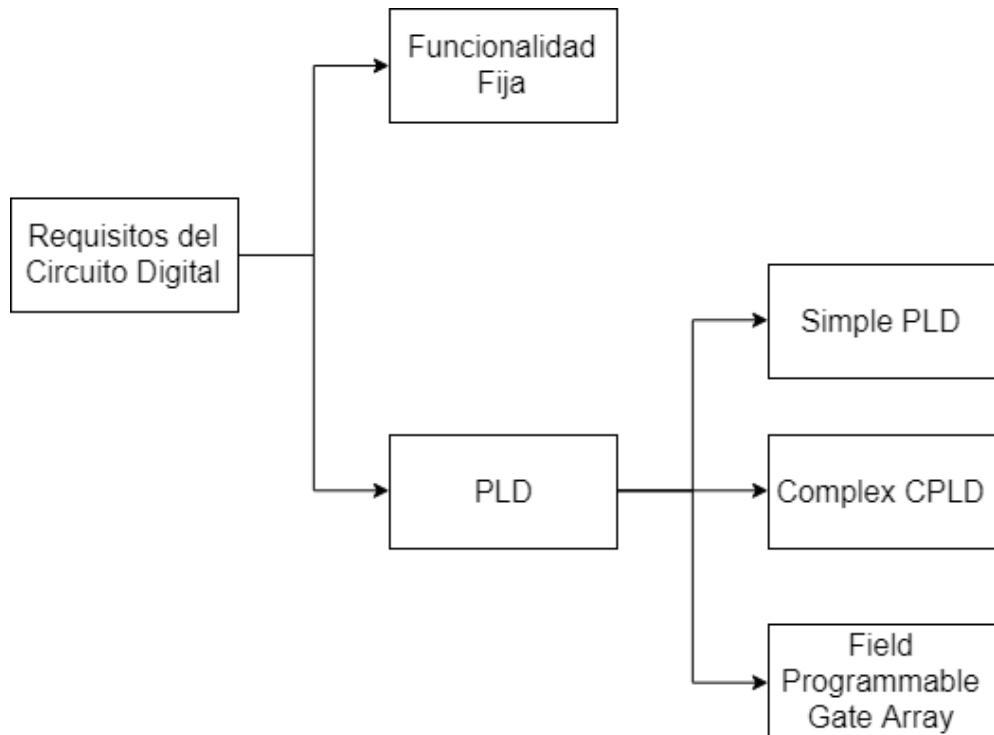


Figura 19. Tecnologías posibles para diseño de circuitos digitales. Nota. Adaptado de “Digital Systems Design with FPGAs and CPLDs”, Grout, Ian, 2008, p. 2, USA: Newnes.

## 2.7. Fabricantes de dispositivos de lógica programable

Hay PLDs disponibles de variedad de fabricantes, cada uno de los cuales ofrece una familia de PLDs basada en SPLD, CPLD o FPGA. También proporcionan las herramientas necesarias para configurar estos PLDs.

La siguiente tabla identifica algunas compañías.

Tabla 1.

Fabricantes de dispositivos lógicos programables

Fabricante	Página web
Xilinx, Inc.	<a href="https://www.xilinx.com/">https://www.xilinx.com/</a>
Altera Corporation	<a href="https://www.altera.com/">https://www.altera.com/</a>
Microsemi Corporation	<a href="https://www.microsemi.com/">https://www.microsemi.com/</a>
Lattice Semiconductor Corporation	<a href="http://www.latticesemi.com/">http://www.latticesemi.com/</a>
QuickLogic Corporation	<a href="https://www.quicklogic.com/">https://www.quicklogic.com/</a>
Cypress Semiconductor Corporation	<a href="http://www.cypress.com/">http://www.cypress.com/</a>

Actualmente (año 2018) el mercado está dominado, principalmente, por Xilinx y Altera (al presente, esta última formando parte de Intel).



Figura 20. FPGA Altera Cyclone IV EP4CE115F23C8N

En este trabajo, vamos a usar una FPGA libre y herramientas libres. Actualmente, las únicas FPGAs libres son las de la familia ICE40 de Lattice. Disponemos de toda la documentación gracias a Clifford Wolf, que hizo ingeniería inversa y creó el proyecto Icestorm. El resto de FPGAs son propietarias, es decir, solo podemos usarlas con el software proporcionado por el fabricante y solo podemos hacer con ellas lo que el fabricante haya decidido que se puede hacer. No están disponibles sus detalles internos con el detalle suficiente, ni el bitstream (veremos más adelante lo que es). Esto hace imposible que alguien que no sea el fabricante cree su propio software para usarla o que pueda sintetizar hardware desde cualquier otra plataforma diferente a la decidida por el fabricante. Algunas placas con FPGAs libres, que además están soportadas por el software que vamos a utilizar nosotros para configurar la FPGA, son estas [4]:

*Tabla 2.*

#### FPGAs libres y soportadas por ICEStudio

Nombre de la placa	FPGA usada	Página web
Lattice Icestick	ICE40HX-1K-TQ144	<a href="http://www.latticesemi.com/icestick">http://www.latticesemi.com/icestick</a>
IceZUM Alhambra	ICE40HX-1K	<a href="https://github.com/FPGAwars/icezum/wiki">https://github.com/FPGAwars/icezum/wiki</a>
Kéfir I	iCE40HX4K-TQ144	<a href="http://fpgalibre.sourceforge.net/Kefir/">http://fpgalibre.sourceforge.net/Kefir/</a>
Nandland Go Board	ICE40HX-1K	<a href="https://www.nandland.com/">https://www.nandland.com/</a>
Icoboard 1.0	ICE40HX-8K	<a href="http://icoboard.org/about-icoboard.html">http://icoboard.org/about-icoboard.html</a>

## 2.8. Metodologías y herramientas de diseño de lógica programable

Para implementar nuestro circuito en un PLD en particular, se requieren las herramientas de diseño apropiadas. Los fabricantes que vimos en la tabla 1, dan sus propias herramientas software de carácter propietario, es decir, no conocemos los detalles internos de su funcionamiento. En este trabajo vamos a usar un software de carácter libre, llamado IceStudio.

Aunque cada herramienta puede diferir de otra en apariencia y forma de funcionamiento en la que el diseñador interactúa con ella, todas tienen un conjunto de características básicas requeridas para crear e implementar diseños digitales.: partimos de un **diseño**. Puede ser un dibujo en un papel o podemos usar CAD (*computer-aided design*). A continuación, bien gráficamente, o bien mediante un lenguaje de descripción de hardware (*HDL, hardware description language*), generamos un fichero de **descripción** hardware, siendo **Verilog** y **VHDL** los lenguajes de descripción más utilizados. Como ejemplo, una analogía: igual que el HTML describe la estructura de una página web, el HDL describe la estructura de un hardware. A continuación, a partir de la descripción se realiza la **síntesis**, que genera un archivo que llamamos “**bitstream**”, que es el que contiene todas las uniones que se tienen que hacer dentro de la FPGA para que aparezca nuestro circuito en ella.

Por último, cargamos este bitstream en la FPGA, es decir, se **configura**. Ya tenemos nuestro circuito [1]. Más en detalle:

**Entrada del diseño:** introducir el diseño en la herramienta usando algún lenguaje de descripción de hardware (VHDL o Verilog) o visualmente (con gráficos).

**Simulación del diseño:** una vez tenemos el diseño del circuito, puede ser simulado para comprobar que funciona como es requerido.

**Síntesis del diseño:** la descripción del hardware (por código o visualmente) se convierte a un conjunto de puertas lógicas e interconexiones llamado “netlist”.

**Emplazado y enrutado (*place and route*):** mapear el diseño sintetizado a los recursos hardware del PLD en concreto. Con esto definimos qué partes del PLD contendrán qué funciones en el diseño y cómo las diferentes partes del PLD se interconectarán. Como cada PLD es diferente, el mapeado es único para cada PLD y la herramienta software debe tener soporte para el PLD en concreto. Todos los valores para los bits de configuración de cada celda de memoria se agrupan en una tira de bits llamada “bitstream”, que se carga desde el exterior:

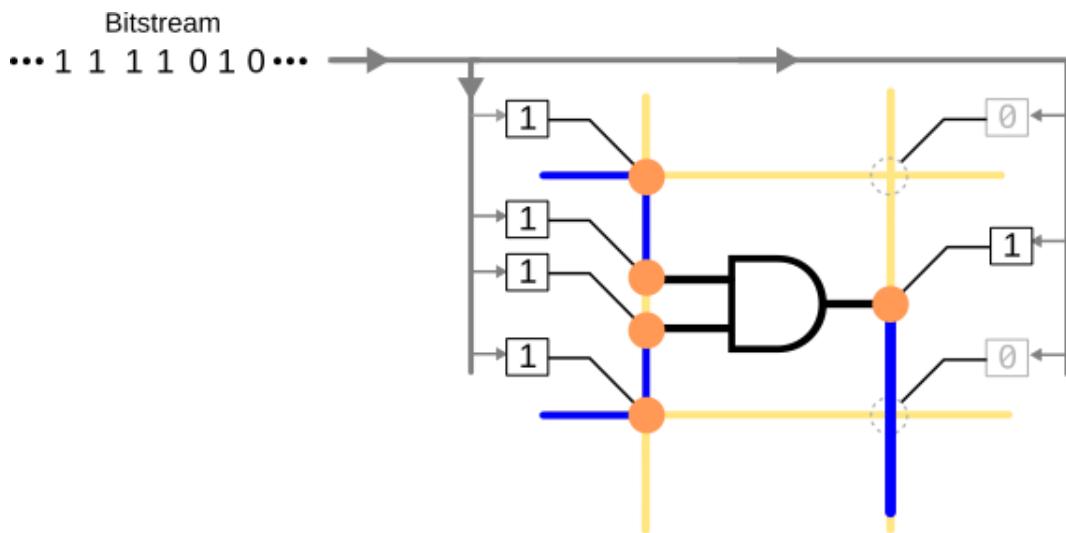


Figura 21. Carga del bitstream en el dispositivo. Nota. Recuperado de “*Introducción*”, de FPGA Wars.  
Recuperado de <http://obijuan.github.io/intro-fpga.html>

El bitstream se transmite por un bus serie de tipo *SPI* (*Serial Peripheral Interface*), bit a bit, configurándose las conexiones del dispositivo [1].

**Extracción de los tiempos de retardo (después del place and route):** toma la información del diseño ya enrutado y extrae los tiempos de retardo debido a las puertas lógicas y a las interconexiones usadas.

**Simulación (después del place and route):** usando los tiempos de retardo, el diseño es simulado de nuevo con estos retardos incluidos para determinar si el diseño todavía funciona correctamente.

**Generación de archivo de configuración:** crea los datos de configuración del PLD.

**Configuración del PLD:** carga los datos de configuración en el PLD y habilita la configuración en el PLD para ser verificar su corrección.

**Interfaz a herramientas externas:** permite a herramientas de tipo *third-party* como herramientas de simulación y síntesis trabajar con las herramientas de diseño principales.

La siguiente figura muestra un esquema simplificado del flujo de diseño típico de una FPGA.

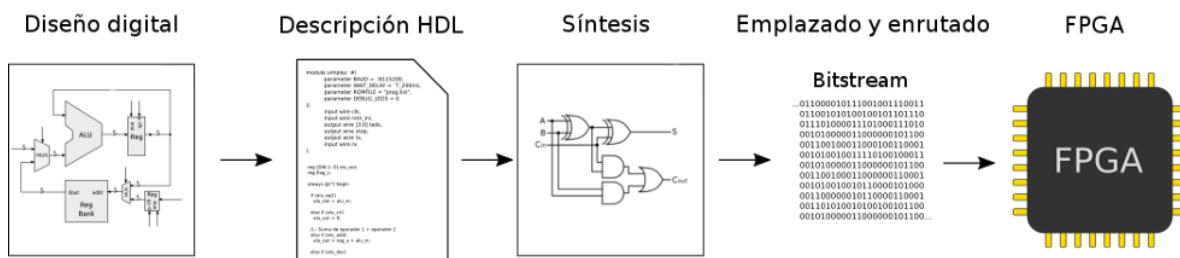


Figura 22. Carga Flujo de trabajo en una FPGA. Nota. Recuperado de “*Introducción*”, de FPGA Wars.  
Recuperado de <http://obijuan.github.io/intro-fpga.html>

## 2.9. JTAG

JTAG es un estándar de la industria creado por la organización *Join Test Action Group* para verificar diseños y probar circuitos después de su fabricación.

Provee a los chips de una interfaz especial llamada interfaz JTAG. Dependiendo de la versión del estándar hay dos, cuatro o cinco pines. Los pins de conexión son: *TDI (Test Data In)*, *TDO (Test Data Out)*, *TCK (Test Clock)*, *TMS (Test Mode Select)* y *TRST (Test Reset, es opcional)*. [16]

No lo vamos a ver en más profundidad en este trabajo.

## 2.10. Estado actual de las FPGAs

A pesar de ser una tecnología disponible desde hace más de treinta años (Xilinx inventó la primera FPGA comercialmente viable en 1985), la difusión de este tipo de circuitos y su accesibilidad al usuario en general es un hecho relativamente reciente. A ello han contribuido múltiples factores, entre los cuales podemos destacar la adquisición de Altera, uno de los mayores fabricantes de FPGAs a nivel mundial, por parte de Intel en 2015. La entrada en escena de este último ha incrementado el nivel de competencia, provocando el lanzamiento de productos basados en FPGA con mayores capacidades y a menor precio.

El CEO de Intel Brian Krzanich resaltó en la *Intel SoC FPGA Developer Forum 2016* la importancia que tiene para Intel el negocio de las FPGAs. Quieren aprovechar la transformación digital. La mayor partida de esta empresa viene de la habilidad de aplicar la ley de Moore a su proceso de diseño y fabricación y a su conocimiento de los sistemas y la arquitectura junto con su experiencia en el sector para hacer crecer el negocio base. Por eso, se han unido, por una parte, su innovación, y por otra, la experiencia en el hardware programable de Altera. Uno de los primeros productos que lanzó fue la *Stratix 10 SoC* (2016) con, entre otras características, silicio de 14 nanómetros, núcleos ARM y una arquitectura *Intel HyperFlex FPGA* con ventajas en rendimiento, eficiencia energética, densidad e integración del sistema respecto a anteriores FPGAs existentes. Quieren usar las FPGAs como un recurso acelerador en sus centros de procesamiento de datos y han apostado muy fuerte por este sector.

Otras empresas como Schneider, empresa dedicada a la fabricación dispositivos IoT, también usan FPGAs, trabajando además con Intel y Altera, usando FPGAs SoC Cyclone 5 en sus plataformas de control embebidas. En el mundo de IOT hay diferentes plataformas de comunicación, como Modbus, PROFINET, Ethernet, etc. En el pasado, esos chips tendrían que haber sido chips ASIC, es decir, chips personalizados para cada uno de estos protocolos. Con *FPGAs SoC* podemos programar lo que queramos. Ahí reside gran parte del poder.

Esta es una tendencia está llegando también a otro grande como Amazon. Podemos ver un ejemplo en los servicios ofrecidos por AWS (Amazon Web Services), en los que el desarrollador cuenta con instancias de ejecución que combinan los servidores tradicionales con hardware reconfigurable de tipo FPGA. A fecha del año 2018, ofrece instancias llamadas F1 de Amazon EC2, que contiene una AMI (Amazon Machine Image), que es como se llama a una configuración determinada para una instancia, basada en el sistema operativo CentOS proporcionada por AWS. Posee herramientas para desarrollar, simular, depurar y compilar para FPGAs, como software *Xilinx Vivado* o *SCDAccel* y HDL como Verilog o VHDL, esto es, un kit de desarrollo de hardware (HDK). Una vez que tenemos el diseño completo para la FPGA, se genera el bitstream junto con varios datos más necesarios propios de Amazon formando lo que se llama AFI (Amazon FPGA Image), que, además, es cifrada para más seguridad. Al final, se carga en la FPGA usando servicios de AWS.

Microsoft fue el pionero en integrar FPGAs en sus servidores. En 2010 comenzó el proyecto Catapult, una iniciativa cuya intención es transformar la computación en la nube personalizando sus servidores a través de CPUs convencionales y una capa de computación extra compuesta por hardware programable, aquí FPGAs. Comenzaron explorando diferentes alternativas y hardware especializado como GPUs, FPGAs y ASICs. En 2015, habilitaron servidores con FPGAs a gran escala en los centros de datos de Bing y en 2016 en Azure, ambos en producción. Esto supuso un aumento del 50% en rendimiento o una reducción del 25% en latencia. Hoy día (2018), casi cada servidor en los centros de datos de Microsoft integra una FPGA de forma distribuida, creando una capa de computación interconectada y configurable que extiende la capa de computación de las CPUs. Además, el proyecto Brainwave está llevando el proyecto Catapult al mundo de la inteligencia artificial. [17]

# Accelerating Large-Scale Services – Bing Search

1,632 Servers with FPGAs Running Bing Page Ranking Service (~30,000 lines of C++)

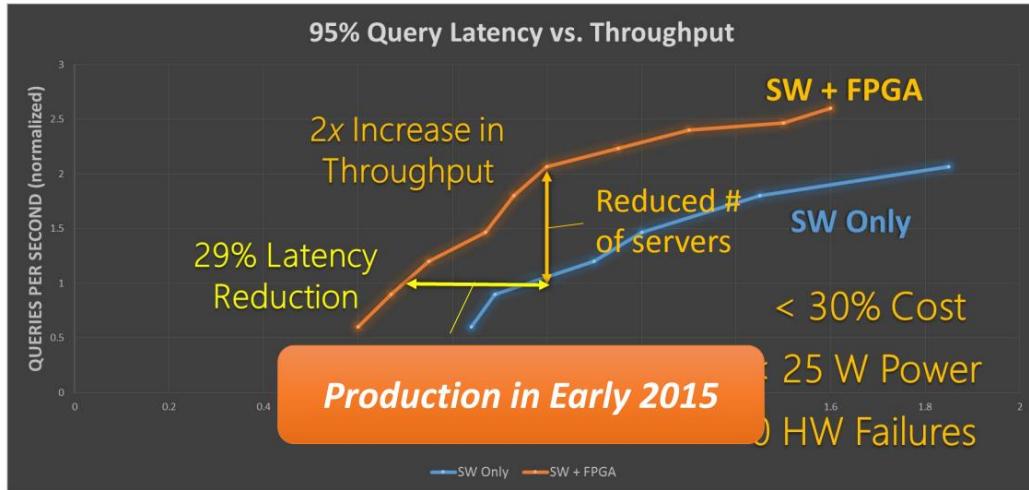


Figura 23. Resultados de las pruebas realizadas en los servidores de Bing con FPGAs antes de lanzarlos a producción. Nota. Recuperado de “Large-Scale Reconfigurable Computing in a Microsoft Datacenter”, de la conferencia HOT CHIPS 26, 2014. Recuperado de <https://www.microsoft.com/en-us/research/wp-content/uploads/2014/06/HC26.12.520-Recon-Fabric-Pulnam-Microsoft-Catapult.pdf>

Pero no ha sido hasta ahora que las FPGAs se están empezando a popularizar entre los usuarios. **Se trata de una tecnología muy cerrada**, rodeada de software propietario, en la que solo se puede usar lo que el fabricante indica en las condiciones que describe. No hay lugar para la innovación por parte de la comunidad. No están publicados los detalles internos de la FPGA, ni del formato de los bitstreams.

Sin embargo, **Clifford Wolf**, hizo **ingeniería inversa** a las FPGAs iCE40 de Lattice y en marzo de 2015 creó el proyecto IceStorm y se liberó el primer conjunto de herramientas que permiten pasar de Verilog a bitstream usando solo herramientas libres. [1]

Todo este movimiento tanto por parte de las empresas, como por parte de los usuarios apunta a un gran futuro de esta tecnología. [3]



# 3. Detalles del proceso

Antes de coger la FPGA y ponernos a trabajar con ella, necesitamos diseñar nuestra ALU. En general, el funcionamiento de nuestra ALU va a ser el siguiente: al introducir los datos con los que operar, se realizarán todas las operaciones a la vez, pero habrá un multiplexor que recoja la salida de cada una de las operaciones y nos permita seleccionar cuáles de esos resultados queremos escoger como salida final, que mostraremos en unos LEDs, por ejemplo.

## 3.1. Etapa 1: diseño de la ALU en Logisim (simulador)

Para comenzar, es recomendable que hagamos primero un **boceto** en papel. Al igual que en software es recomendable darle una pensada y hacerse un diseño de cómo se va a estructurar nuestro código: qué clases va a tener, qué tipo de relación va a haber entre ellas (herencia, por ejemplo), qué atributos y métodos va a tener, qué tipo de patrones de diseño vamos a aplicar, etc., en hardware también es útil pensar antes de ponerse manos a la obra. Vamos a pensar qué componentes vamos a necesitar (puertas lógicas, módulos, cables, etc.) y cómo los vamos a conectar. Además, es una tarea más visual, ya que trabajamos con puertas lógicas y módulos con funcionalidades específicas (por ejemplo, un sumador) ya hechos.

A la hora de introducirlos en un simulador, tenemos múltiples opciones respecto al software a usar. Logisim es uno de ellos, es libre, lleva en desarrollo desde hace varios años, es conocido en Internet y, por lo tanto, es más probable encontrar soporte y ayuda a cualquier problema que se nos presente con él. Con un software de este tipo tenemos la oportunidad de simular el funcionamiento de los componentes en nuestro ordenador.

Nota: los diseños de los distintos componentes se adjuntan junto con este documento.

## Descripción general

Este circuito es una Unidad Aritmético Lógica (ALU, Arithmetic Logic Unit) que, en total, puede llevar a cabo 11 operaciones, entre operaciones lógicas y aritméticas.

- 4 operaciones aritméticas suma, resta, incremento y decremento.
- 7 operaciones lógicas desplazamiento a izquierda y derecha, rotación a izquierda y derecha, AND, OR y XOR.

## Símbolo lógico

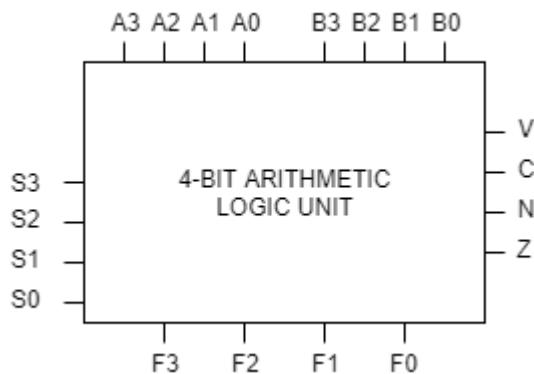
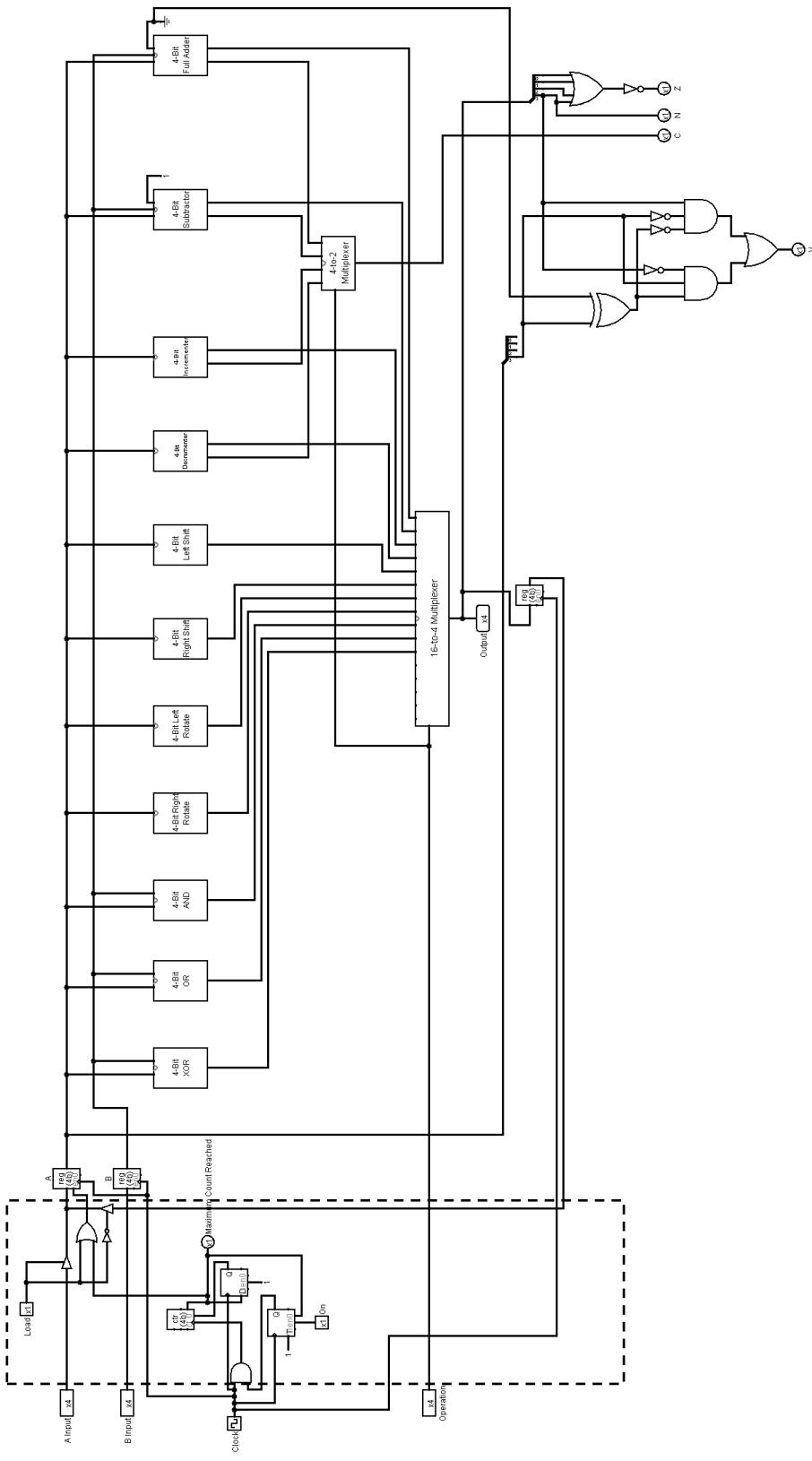


Figure 24. Símbolo lógico que representa nuestra ALU.

## Nombre de los pines

Tabla 3.  
Nombre de los pines de nuestra ALU

Nombre del pin	Descripción
A3...A0	Operando de entrada (Activo en HIGH)
B3...B0	Operando de entrada (Activo en HIGH)
S3...S0	Selección de la operación
F3...F0	Resultado de la operación
V	Overflow: indicativo de overflow en el resultado
C	Acarreo: indicativo de acarreo en el resultado
N	Negativo: indicativo de resultado con signo negativo
Z	Zero: indicativo de resultado igual a cero



**Diagrama lógico**

Figura 25. Diagrama lógico de nuestra ALU. La sección del circuito contenida dentro del recuadro con borde punteado no pertenece a la ALU como tal, sino a la lógica de control.

## Descripción funcional

Controlada por las cuatro entradas de selección (F3...F0), esta ALU puede realizar hasta 11 operaciones en señal activa en HIGH. La tabla de funciones lista estas operaciones.

Cuando se introducen las entradas A y B como operandos, la entrada S3...S0 como selector de la operación y las señales de control permiten su ejecución, internamente todas las operaciones se llevan a cabo en paralelo simultáneamente. Posteriormente, un multiplexor es el encargado de dejar pasar hasta la salida, el resultado de la operación asociada a la entrada S3...S0.

## Tabla de funciones

Tabla de funciones					
Entradas de selección de la operación				Entradas activas en HIGH & salidas	Descripción
S3	S2	S1	S0		
L	L	L	L	A + B	Suma aritmética de A y B
L	L	L	H	A - B	Resta aritmética de A y B
L	L	H	L	A + 1	Incremento en 1 unidad de A
L	L	H	H	A - 1	Decremento en 1 unidad de A
L	H	L	L	BSL(A, 1)	Desplazamiento de bits a la izquierda (Bit Shift Left) de A 1 unidad
L	H	L	H	BSR(A, 1)	Desplazamiento de bits a la derecha (Bit Shift Right) de A 1 unidad
L	H	H	L	BROTL(A, 1)	Rotación de bits a la izquierda (Bit Rotation Left) de A 1 unidad
L	H	H	H	BROTR(A, 1)	Rotación de bits a la derecha (Bit Rotation Right) de A 1 unidad
H	L	L	L	A AND B	AND lógico de A y B
H	L	L	H	A OR B	OR lógico de A y B
H	L	H	L	A XOR B	XOR lógico de A y B
H	L	H	H	NC	No conectado
H	H	L	L	NC	No conectado
H	H	L	H	NC	No conectado
H	H	H	L	NC	No conectado
H	H	H	H	NC	No conectado

### 3.1.1. Operaciones aritméticas de 4 bits

#### 3.1.1.1. Sumador de 4 bits (4-Bit Full Adder)

La suma es la operación más básica de las operaciones aritméticas, por lo que, si queremos construir una ALU, primero debemos saber cómo construir algo que sume dos números juntos.

En binario sumamos de la siguiente manera:

+	0	1
0	0	1
1	1	10

Si reescribimos la tabla con ceros delante:

+	0	1
0	00	01
1	01	10

Viéndolo así, el resultado de sumar un par de números binarios es 2 bits, llamados **bit de suma (sum bit)** y **bit de acarreo (carry bit)**. Ahora podemos dividir la tabla de suma binaria en dos tablas, la primera para el bit de suma y la segunda para el bit de acarreo:

+	0	1
(sum)	0	1
0	0	1
1	1	0

+	0	1
(carry)	0	0
0	0	0
1	0	1

Si nos fijamos, el bit de suma puede implementarse como una puerta XOR:

+	0	1
(sum)	0	1
0	0	1
1	1	0



XOR	0	1
0	0	1
1	1	0

Si nos fijamos, el carry puede implementarse como una puerta AND:

+	0	1
<b>(carry)</b>	0	1
0	0	0
1	0	1

→

<b>AND</b>	0	1
0	0	0
1	0	1

Por lo tanto, ya tenemos un sumador de 1 bit, lo que conocemos como un *Half Adder*.

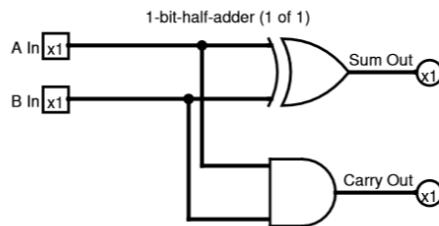


Figura 24. Digráma lógico de nuestro 1-Bit Half Adder de 1 bit (1-Bit Half Adder)

Si no queremos dibujar una y otra vez el circuito, podemos usar un gráfico (una caja), como se muestra a continuación:



Figura 25. Símbolo lógico de nuestro Half-Adder de 1 bit

Se llama Half Adder por una razón. Efectivamente, es capaz de sumar un par de bits y de darnos su correspondiente bit de suma y el de acarreo. Pero donde falla es al sumar un posible bit de acarreo proveniente de una suma previa (el tradicional bit que nos “llevamos”). Es decir, no es capaz de sumar tres bits. Para sumar tres bits podemos usar un Half Adder y una puerta OR, obteniendo lo que se llama un *Full Adder*, como se muestra en la siguiente figura:

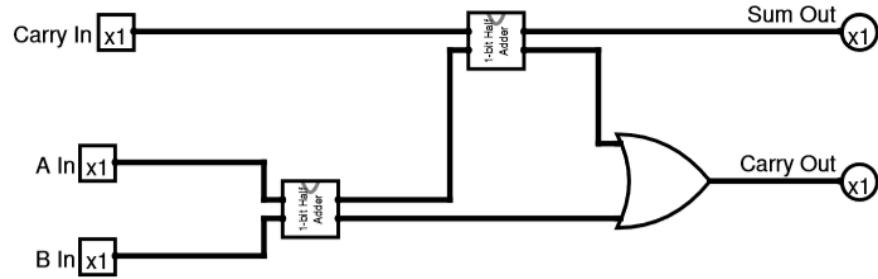


Figura 26. Full Adder de 1 bit (1-Bit Full Adder)

Ahora, imaginemos que queremos realizar la siguiente suma:

$$\begin{array}{r}
 0010 \\
 + 0101 \\
 \hline
 \end{array}$$

Podemos combinar 4 Full Adders para sumar números de 4 cifras binarias. La primera columna (la de las unidades, si estuviéramos en el sistema decimal) no recibe bit de acarreo, por lo que la conectamos a tierra o le introducimos un bit a 0. Para la siguiente columna, el bit de acarreo de salida del primer Full Adder se conecta como bit de acarreo de entrada al siguiente Full Adder. Y así con todas las demás. Finalmente, el último bit de acarreo de salida se conecta a la salida (un LED, por ejemplo).

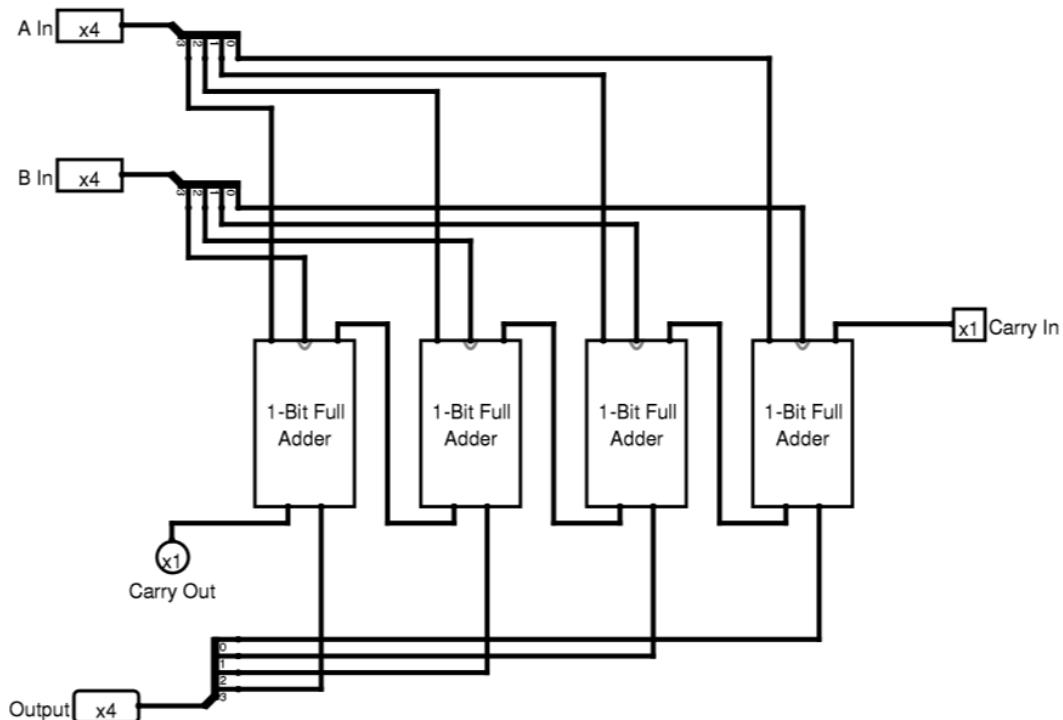


Figura 27. Sumador de 4 bits (4-Bit Full Adder)

### 3.1.1.2. Restador de 4 bits (4-Bit Subtractor)

Para hacer el restador, usaremos el Full Adder anterior con dos diferencias: la entrada Carry In está a 1 (veremos por qué) y la entrada B es invertida. La razón radica en que vamos a utilizar el concepto del **complemento a dos**: si queremos restar  $A - B$ , A lo dejamos como está, pero B lo invertimos (en binario) y le sumamos una unidad, es decir, hemos calculado su complemento a dos. Eso sí, debemos tener en cuenta que el resultado de la operación también va a estar representado en complemento a dos y debemos interpretarlo como tal.

Con lo cual, lo único que necesitamos son dos entradas: A y B. La entrada A puede permanecer inalterada, pero la entrada B la invertimos usando puertas NOT y le sumamos 1 aprovechándonos de la primera entrada de acarreo, que estará a 1:

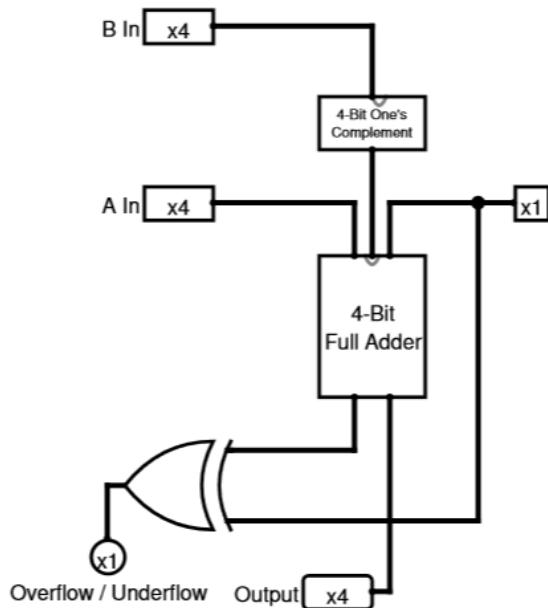


Figura 28. Restador de 4 bits (4-bit Subtractor)

### 3.1.1.3. Incrementador de 4 bits (4-Bit Incrementer)

El incrementador está compuesto de Half Adders de 1 bit, y lo único especial es que la entrada B es un bit a 1, ya que esa va a ser siempre su función: sumar una unidad al dato de entrada.

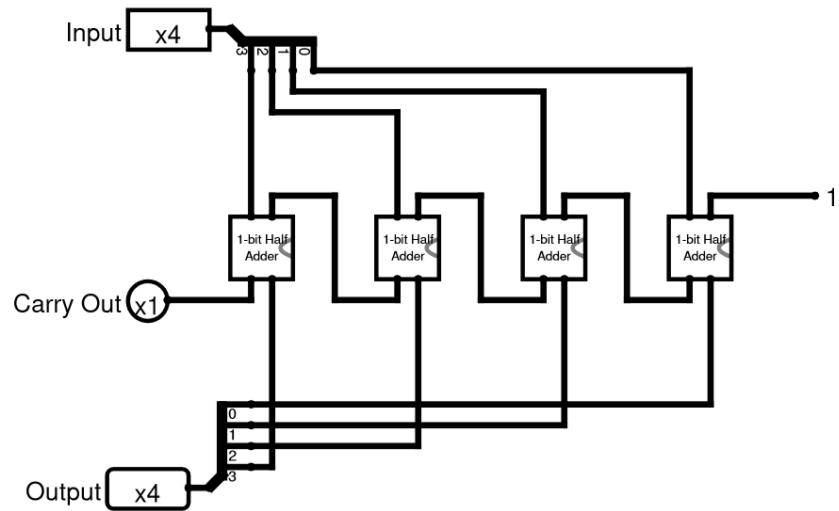


Figura 29. Incrementador de 4 bits (4-Bit Incrementer)

### 3.1.1.4. Decrementador de 4 bits (4-Bit Decrementer)

El decrementador es un sumador de 4 bits con el Carry In a 0 y todos los bits de la entrada B a 1. Porque restar 1 es lo mismo que sumar 1111.

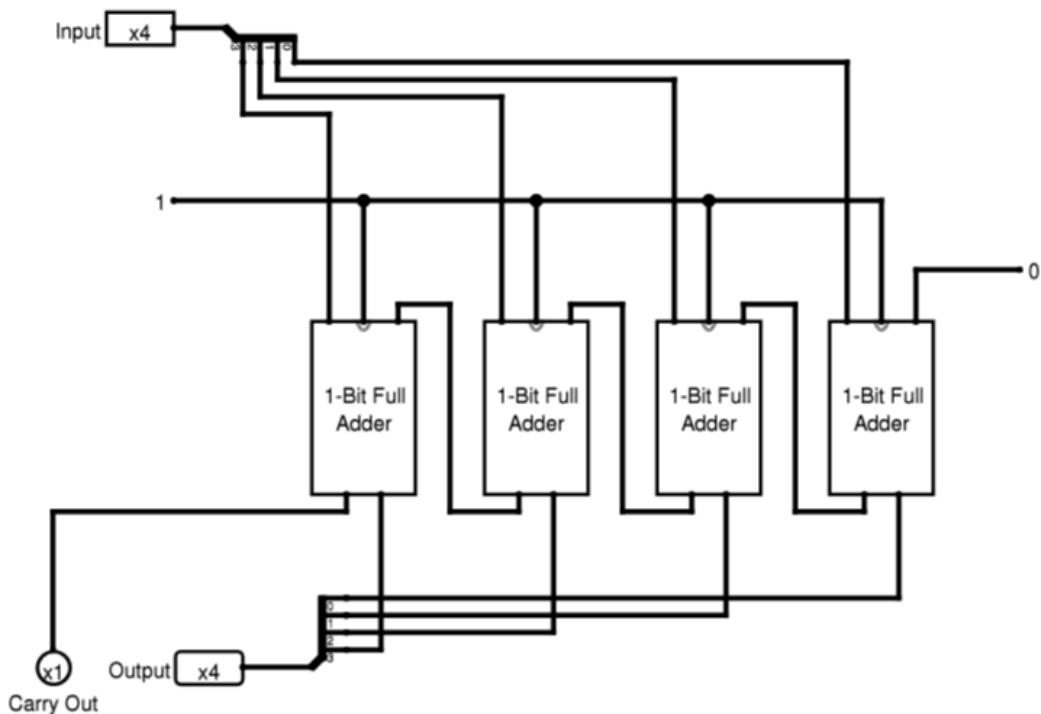


Figura 30. Decrementador de 4 bits (4-Bit Decrementer)

### 3.1.2. Operaciones lógicas de 4 bits

Implementar las operaciones lógicas nos va a resultar fácil porque podemos mapearlas directamente a puertas lógicas.

#### 3.1.2.1. AND de 4 bits (4-Bit AND)

Usamos 4 puertas AND, cada una recibiendo un bit de cada entrada.

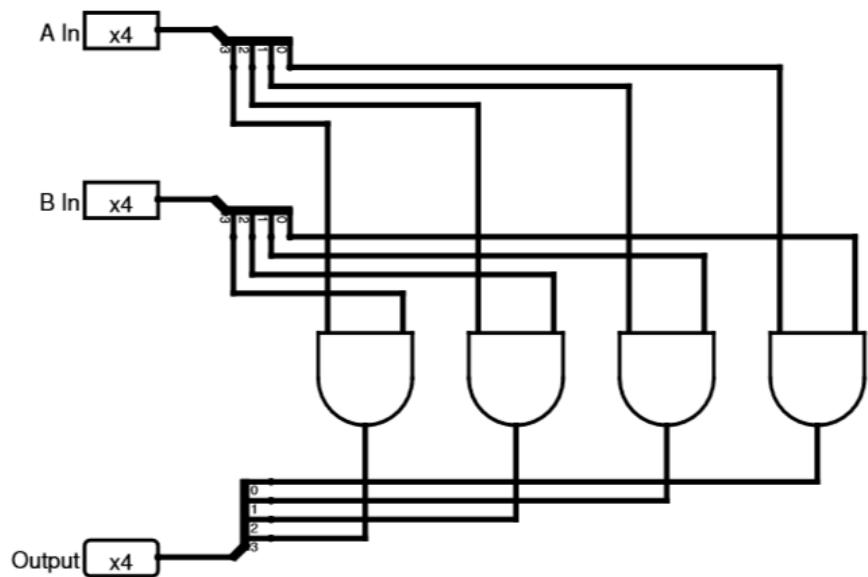


Figura 31. AND de 4 bits (4-Bit AND)

#### 3.1.2.2. OR de 4 bits (4-Bit OR)

Usamos 4 puertas OR, cada una recibiendo un bit de cada entrada.

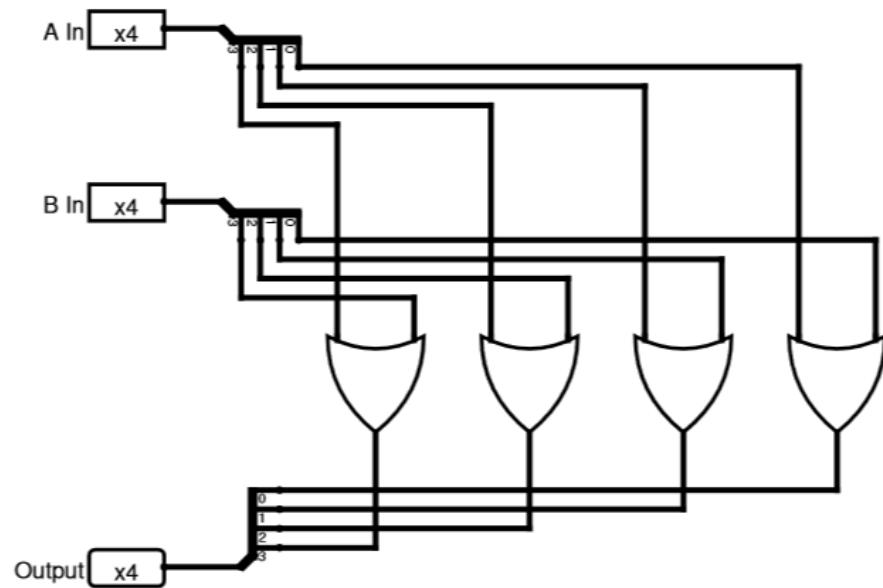


Figura 32. OR de 4 bits (4-Bit AND)

### 3.1.2.3. XOR de 4 bits (4-Bit OR)

Usamos 4 puertas XOR, cada una recibiendo un bit de cada entrada.

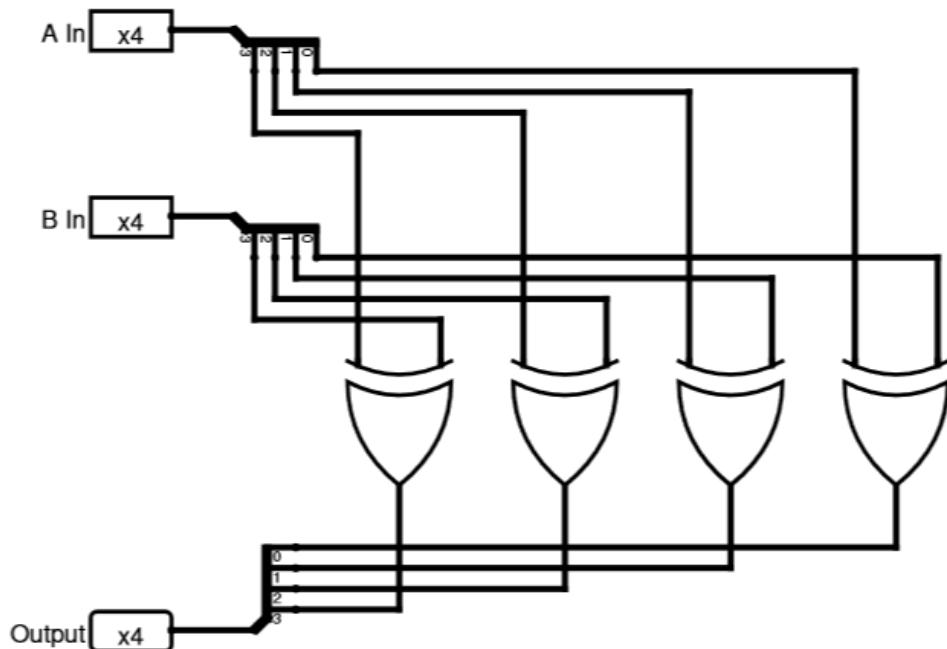


Figura 33. XOR de 4 bits (4-Bit OR)

### 3.1.2.4. Complemento a uno de 4 bits (4-Bit One's Complement)

Usamos 4 puertas NOT, cada una recibiendo un bit de la entrada.

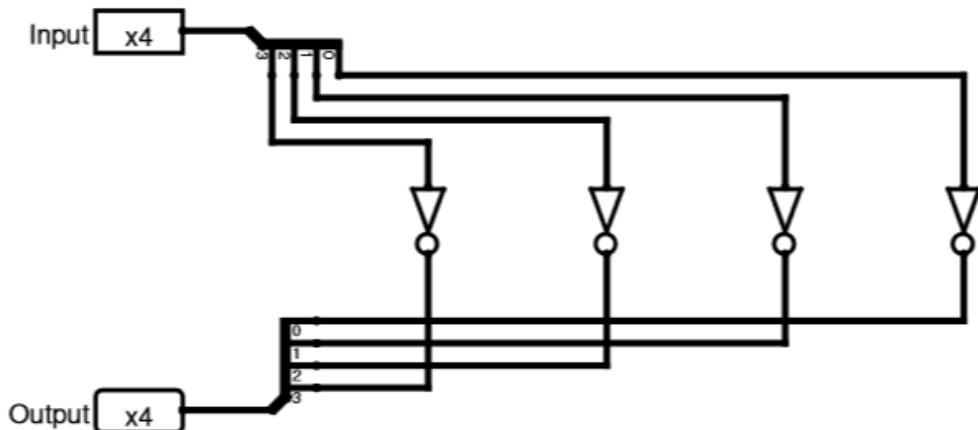


Figura 34. Complemento a uno de 4 bits (4-Bit One's Complement)

### 3.1.3. Operaciones de desplazamiento de 4 bits

Otro circuito, llamado de desplazamiento o *shifter*, es muy importante. Simplemente copian la entrada que reciben en las salidas, únicamente haciendo una ligera permutación de los valores, normalmente desplazando la salida de un bit hacia la derecha o hacia la izquierda.

#### 3.1.3.1. Desplazamiento lógico a la izquierda de 4 bits (4-Bit Left Shift)

Insertar 0s por la izquierda y el bit más significativo (*Most Significant Bit, MSB*) se pierde. Por ejemplo: si recibimos 1001, la salida es 0100 (hemos insertado un 0 a la izquierda y perdido el bit 1 original de la derecha). [18]

Podemos ver que la entrada Right Shift está a 0 y la entrada Left Shift está a 1.

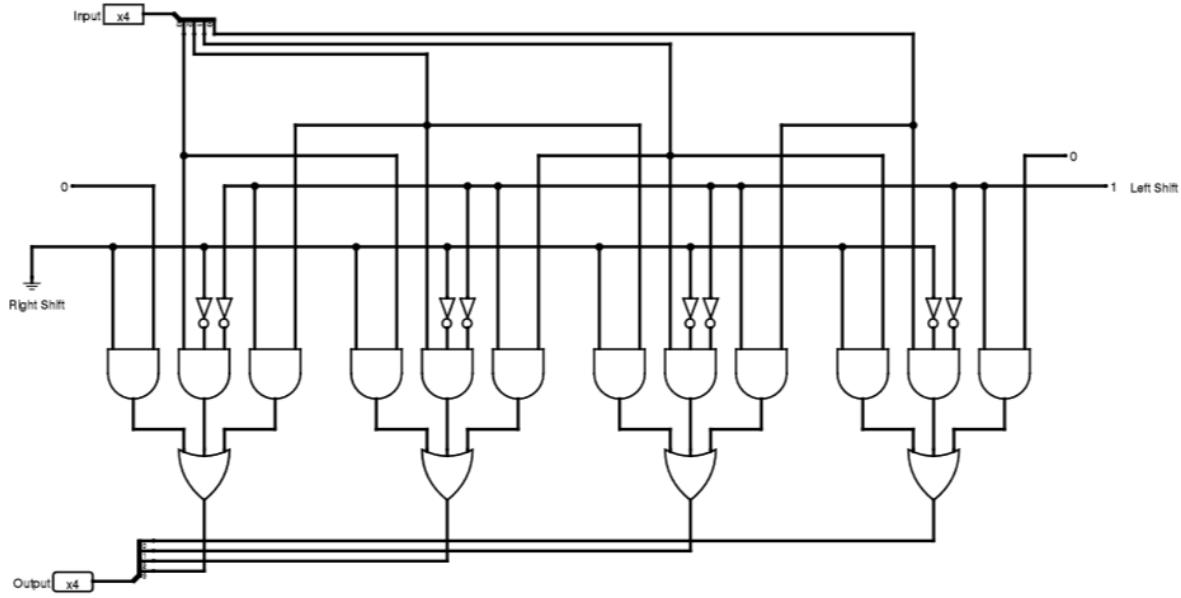


Figura 35. Desplazamiento a la izquierda de 4 bits (4-Bit Left Shift)

### 3.1.3.2. Desplazamiento lógico a la derecha de 4 bits (4-Bit Right Shift)

Insertar 0s por la derecha y el bit menos significativo (*Least Significant Bit, LSB*) se pierde. Por ejemplo: si recibimos 1001, la salida es 0010 (hemos insertado un 0 a la derecha y perdido el bit a 1 original de la izquierda).

Podemos ver que la entrada Left Shift está a 0 y la entrada Rigth Shift a 1.

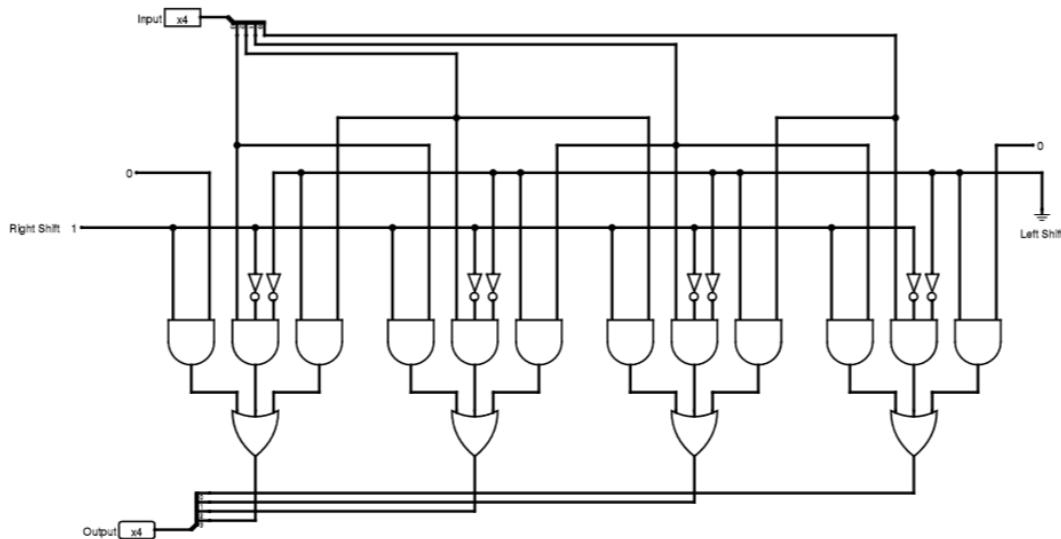


Figura 36. Desplazamiento a la derecha de 4 bits (4-Bit Right Shift)

### 3.1.3.3. Rotación a la izquierda de 4 bits (4-Bit Left Rotate)

Se coge el bit de la izquierda y se pone al principio por la derecha. Es un desplazamiento a la izquierda con la diferencia de que el MSB se pone a la derecha del LSB, pasando él mismo a ser el nuevo LSB. Por ejemplo: si recibimos 1001, la salida sería 0011.

Podemos ver que la entrada Right Shift está a 0 y la entrada Left Shift a 1.

Se diferencia del circuito de desplazamiento lógico a la derecha en que el MSB es el resultado de aplicar un OR al MSB del desplazamiento lógico y el LSB original. Si tenía 1001 y hago un desplazamiento lógico a la izquierda obtengo 0010. Pero si cogemos el MSB actual, 0, y le aplicamos un OR con el LSB original, es decir, 1, obtenemos la salida del LSB final, que es 1. Si en su lugar hubiéramos recibido un 1101 y hago un desplazamiento lógico a la izquierda obtengo 1010. Pero si cogemos el MSB actual, 1, y le aplicamos un OR con el LSB original, es decir, 1, obtenemos la salida del LSB final, que es 1.

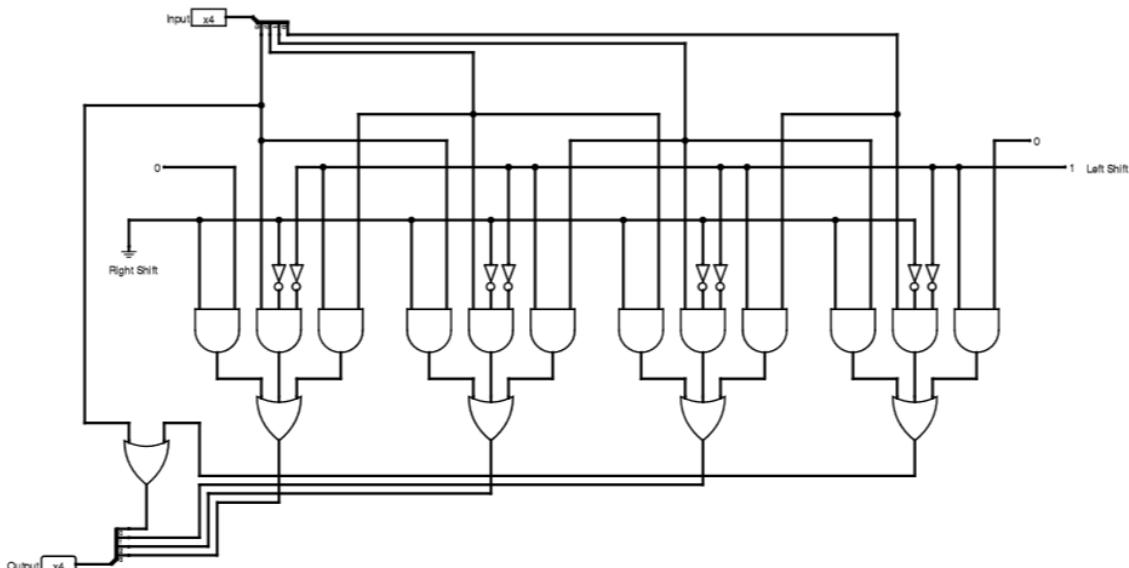


Figura 37. Rotación a la izquierda de 4 bits (4-Bit Left Rotate)

### 3.1.3.4. Rotación a la derecha de 4 bits (4-Bit Right Rotate)

Se coge el bit de la derecha y se pone al principio por la izquierda. Es un desplazamiento a la derecha con la diferencia de que el LSB se pone a la izquierda del MSB, pasando él mismo a ser el nuevo MSB. Por ejemplo: si recibimos 1001, la salida sería 1100.

Podemos ver que la entrada Left Shift está a 0 y la entrada Rigth Shift a 1.

Funciona de manera similar a la rotación a la izquierda de 4 bits.

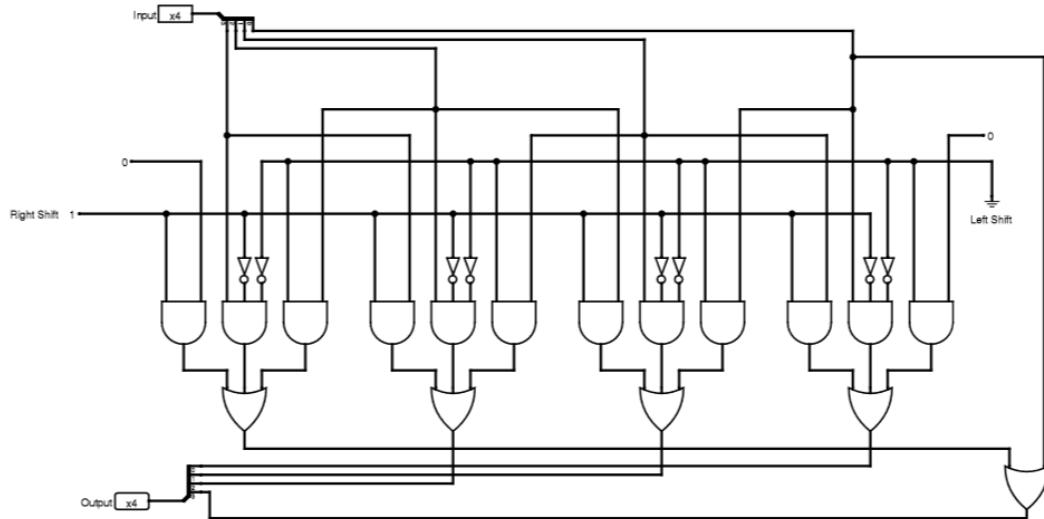


Figura 38. Rotación a la derecha de 4 bits (4-Bit Right Rotate)

### 3.1.4. Circuitos útiles

#### 3.1.4.1. Multiplexor de 4 a 2 (4-To-2 Multiplexer)

Buscamos un circuito que reciba un valor por entrada y nos devuelva un resultado dependiendo de ese valor, actuando como un selector de datos. El valor de entrada representa una de las operaciones que puede llevar a cabo la ALU. Este multiplexor nos servirá para seleccionar uno de los posibles acarreos (el de la suma, la resta, el incremento o el decremento). Está nombrado “de 4 a 2” porque es capaz de seleccionar 4 posibles valores a través de una entrada de 2 bits. En este caso, la entrada será de 2 bits. Se puede implementar con puertas AND:

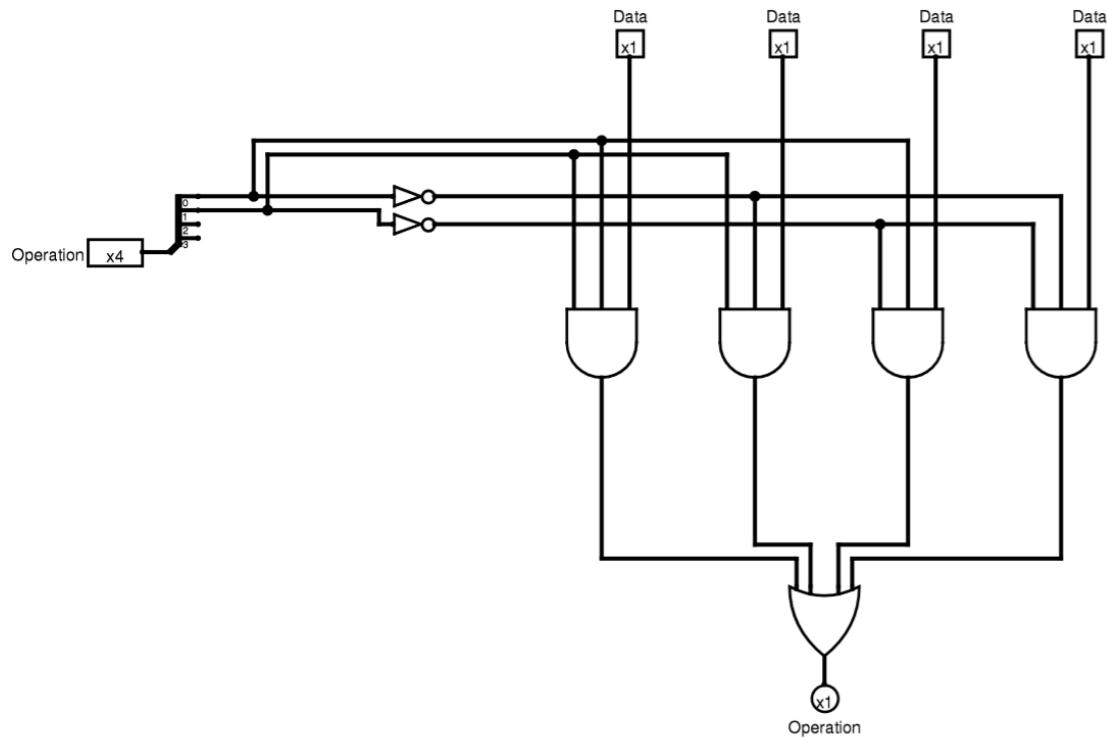


Figura 39. Multiplexor de 4 a 4 (4-To-4 Multiplexer)

La tabla de verdad es la siguiente:

Entrada		Salida
S1	S0	Q
0	0	D0
0	1	D1
1	0	D2
1	1	D3

### 3.1.4.2. Multiplexor de 16 a 4 (16-To-4 Multiplexer)

Sigue el mismo principio que el multiplexor anterior, con la diferencia de que, en este caso, contamos con una entrada de 4 bits y una salida. El valor de entrada representa una de las operaciones que puede llevar a cabo la ALU. El valor de salida es el resultado de aplicar la operación seleccionada sobre las entradas A y B. Está nombrado “de 16 a 4” porque es capaz de seleccionar 16 posibles valores a través de una entrada de 4 bits.

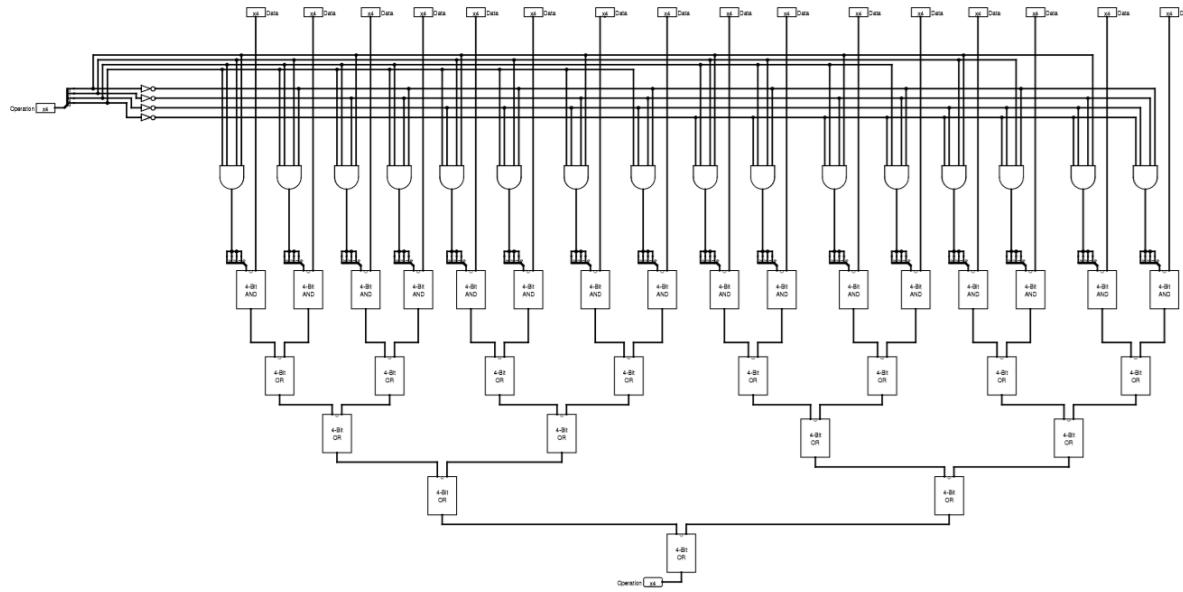


Figura 40. Multiplexor de 16 a 4 (16-To-4 Multiplexer)

La tabla de verdad es la siguiente:

Entrada					Salida
S0	S1	S2	S3	Q	
<b>0</b>	0	0	0	D0	
<b>0</b>	0	0	1	D1	
<b>0</b>	0	1	0	D2	
<b>0</b>	0	1	1	D3	
<b>0</b>	1	0	0	D4	
<b>0</b>	1	0	1	D5	
<b>0</b>	1	1	0	D6	
<b>0</b>	1	1	1	D7	
<b>1</b>	0	0	0	D9	
<b>1</b>	0	0	1	D8	
<b>1</b>	0	1	0	D10	
<b>1</b>	0	1	1	D11	
<b>1</b>	1	0	0	D12	
<b>1</b>	1	0	1	D13	
<b>1</b>	1	1	0	D14	
<b>1</b>	1	1	1	D15	

### 3.1.5. Resultado final: ALU de 4 bits (4-Bit ALU)

Asignamos las entradas A y B a cada una de los submódulos creados anteriormente (el incrementador, decrementador y los módulos de lógica de desplazamiento solo operan sobre el registro A). Conectamos las salidas al multiplexor 16 a 4. Conectamos también los cuatro posibles acarreos al multiplexor 4 a 2. Conectamos también al circuito general la lógica de flags. Finalmente, conectamos la lógica de control. Tras estos pasos, obtenemos el siguiente circuito:

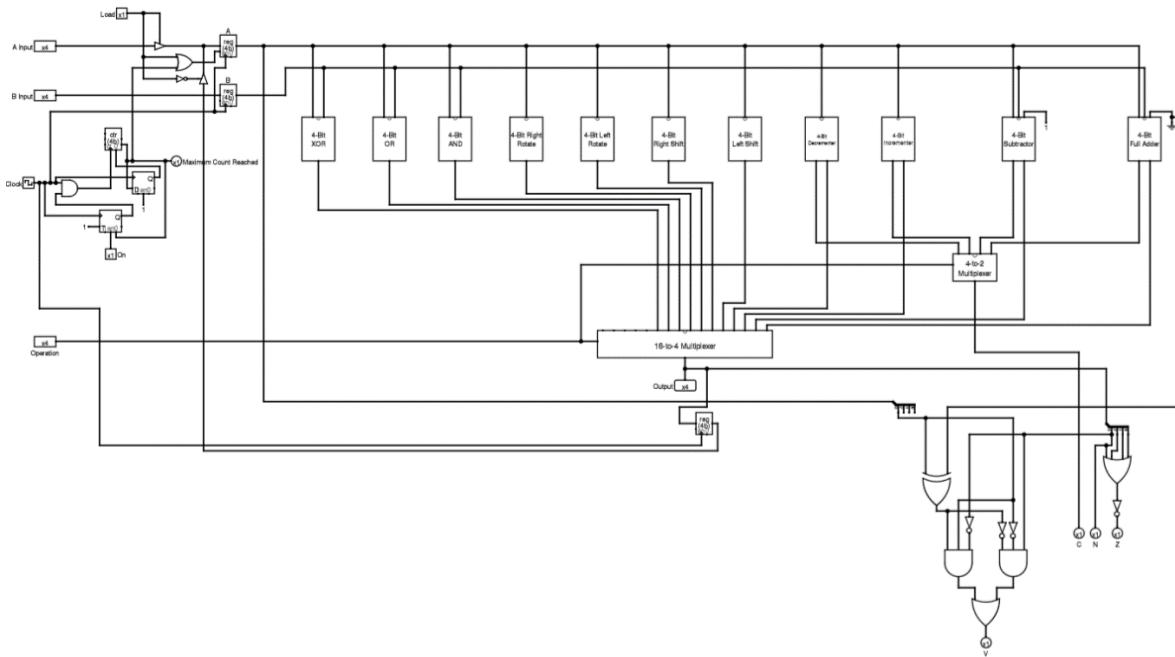


Figura 41. ALU de 4 bits (4-Bit ALU)

#### Instrucciones de uso

1. Introducir un valor en la entrada A
2. Introducir un valor en la entrada B
3. Introducir un valor en la entrada Operation
4. Poner Load a 1 para habilitar el paso de A al registro acumulador
5. Poner On a 1 para comenzar a funcionar

Ahora, en la primera subida de reloj, se cargan los registros A y B.

6. Devolver Load a 0 antes del siguiente ciclo de reloj para habilitar el paso del resultado al acumulador
7. Poner On a 0 para que el registro tipo T (toggle), que controla cuándo para el reloj, no cambie constantemente
8. Dejar pasar ciclos de reloj hasta que se haya actualizado el acumulador
9. Ir al paso 1 para operar con un nuevo valor de A introducido “a mano” o al 3 para continuar operando con lo que hay almacenado en el acumulador

Load sirve de parámetro de entrada a las puertas de estado que controlan qué valor se puede almacenar en el registro A. Las puertas triestado son puertas lógicas con una entrada y una salida ordinarias, con el añadido de una segunda entrada “enable” habilitadora, de forma que si enable está a 1, la salida es igual a la entrada, mientras que si enable está a 0, la salida queda en un estado de “alta impedancia” o z (Hi-Z, high impedance), quedando deshabilitada. La entrada On es la que controla el registro tipo T, que a la vez controla cuándo funciona el reloj y cuándo no. En este caso, cada vez que el contador llega al máximo valor especificado, activa la señal “Clear” del registro toggle, parando el reloj (la señal On estaría ya desactivada, como se indica en el paso 7 de las instrucciones anteriores) y esperando que el usuario decida introducir un nuevo valor en A y/o B o continuar operando con el valor que hay en el acumulador.

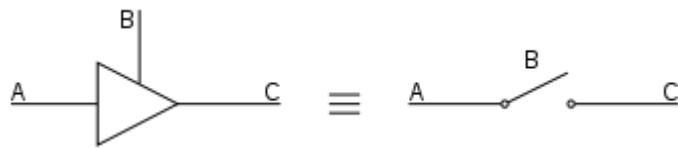


Figura 42. Podemos ver una puerta triestado como un interruptor que activa o desactiva la salida. Nota.  
Recuperado de [https://es.wikipedia.org/wiki/Buffer\\_triestado](https://es.wikipedia.org/wiki/Buffer_triestado)

### 3.2. Etapa 2: transferencia del diseño en Logisim a IceStudio para su implementación en la FPGA

#### 3.2.1. Pila de herramientas libres

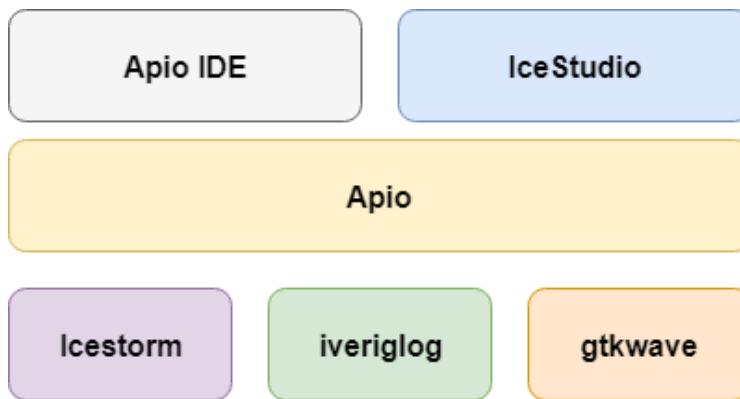


Figura 43. Pila de herramientas libres. Nota. Adaptado de "OSHWDem 2016: Charla FPGAs Libres - Juan González (Obijuan)". Adaptado de <https://www.youtube.com/watch?v=XWC1B7UKv98>

El Proyecto IceStorm es un proyecto creado Por Clifford Wolf cuyo objetivo es proporcionar las herramientas **libres** necesarias para realizar el ciclo completo de trabajo con FPGA, de manera que

no dependamos de ningún fabricante. Wolf fue capaz de hacer ingeniería inversa a unos modelos de FPGAs de Lattice para descifrar el formato del bitstream y documentarlo por completo.

Apio es una herramienta programada en Python que engloba (actuando como un *wrapper*) otras herramientas destinadas a la verificación, sintetización, simulación y carga de diseños en Verilog (el único HDL soportado de forma estable por el proyecto IceStorm). Por ejemplo, *iverilog* sintetiza generando netlists y simula y *gtkwave* visualiza esa simulación.

También sea apoyan sobre Apio dos herramientas: IceStudio y ApioIDE. IceStudio es un editor visual para FPGAs libres, permite diseñar circuitos y cargarlos en la FPGA de una forma gráfica (es como el Scratch del hardware), mientras que Apio IDE es un editor para lenguajes de descripción de hardware (HDLs) basado en Atom con la que podemos diseñar hardware mediante código, sintetizar y simular, teniendo todo a mano.

### 3.2.1.1. IceStudio

Ahora podemos empezar a introducir nuestro diseño en IceStudio. El funcionamiento es simple: elegimos los componentes desde los desplegables del menú superior y los posicionamos sobre el canvas que se extiende en la ventana del programa. Para conectarlos con otros componentes basta con arrastrar una línea desde un componente hasta otro.

IceStudio también nos permite crear submódulos. Estos submódulos se almacenan en un archivo común de IceStudio en formato *.ice*, pero requieren llevar a cabo un paso importante: establecer que las salidas no son ningún pin de la FPGA. Esto lo haremos al importar el submódulo en el módulo que lo contiene, para que la salida pueda ser mapeada a un pin real de la FPGA que estemos usando. Para ello hacemos doble click sobre la salida y marcamos “FPGA pin”, como se muestra en la siguiente secuencia de imágenes:



Figura 44. Configurando el diseño en IceStudio.

El software se basa en tecnologías web. En su primera versión, el framework principal estaba escrito en ES5 sobre AngularJS (que simplifica el desarrollo HTML/JS) y NW.js (que permite convertir aplicaciones web en nativas gracias a Node.js). Además, usaba los componentes JointJS

para los gráficos (basado en BackboneJS), AlertifyJS para las notificaciones (basado en jQuery), ACE como editor de texto y múltiples paquetes de Node.js.

El diseño que creamos anteriormente en Logisim quedaría de la siguiente manera en IceStudio:

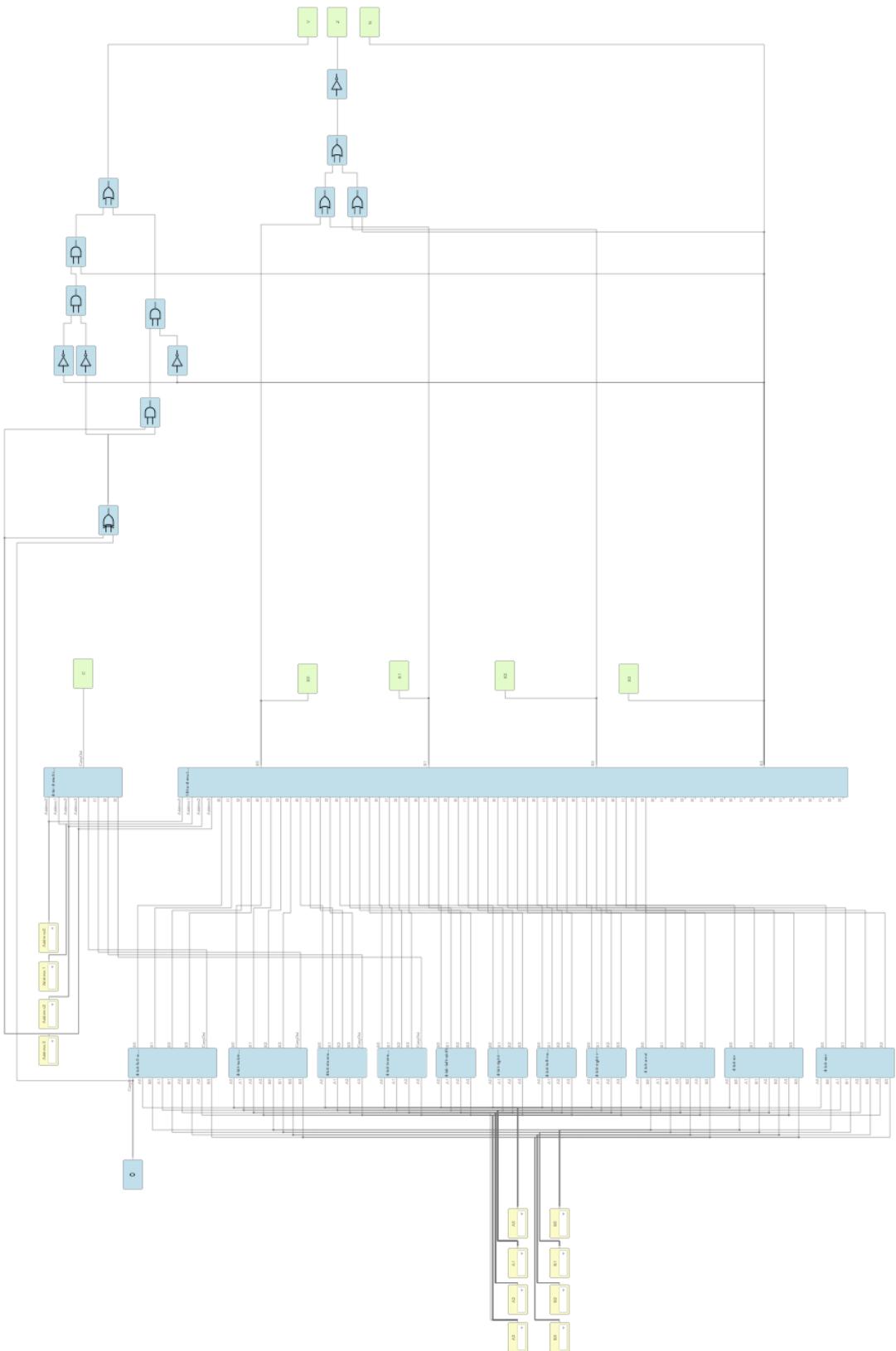


Figura 45. Diseño de la ALU en IceStudio. Listo para ser cargado en la FPGA.

Si “empaquetamos” en un bloque, obtendríamos el siguiente diagrama:

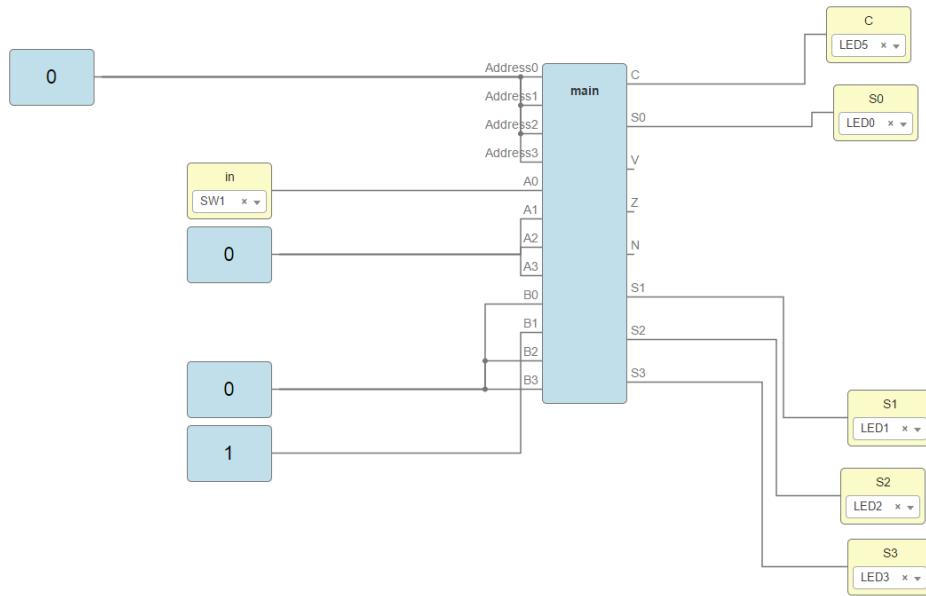


Figura 46. Diagrama de nuestra ALU una vez encapsulada. Un ejemplo en el que la entrada Operation tiene el valor 0000, realizando la suma de A, que vale X000 (X es el valor que tome el pulsador “SW1” de la placa), y B, que vale 0010. El resultado de 4 bits y el acarreo se muestran por los LEDs LED0...LED3 y LED5, respectivamente

## Etapa 3: montaje del sistema y pruebas en la FPGA

### 3.2.2. IceZUM Alhambra

Nosotros trabajaremos con la placa IceZUM Alhambra. Tiene las siguientes características:

- FPGA iCE40HX1K-TQ144 de Lattice Semiconductors
- Hardware libre
- Compatible con las herramientas del Proyecto IceStorm, por Clifford Wolf
- Multiplataforma: Linux / Mac / Windows
- Apariencia similar a la de Arduino: un pinout similar al de Arduino / BQ Zum.
- Compatible con la mayoría de shields de Arduino / BQ Zum
- Oscilador de 12 MHZ MEMS
- Interruptor ON/OFF
- Voltaje de entrada: 6 - 17v
- Corriente de entrada máxima: 3A
- 20 pines de Entrada/Salida
- 8 pines de Entrada/Salida de 3.3V

- Conector micro USB de tipo B para configurar la FPGA desde el PC
- Módulo USB FTDI 2232H que permite la configuración de la FPGA
- Interfaz UART al PC
- Pulsador de reinicio
- 8 LEDs de uso general
- 2 pulsadores de uso general
- LEDs TX/RX
- LED indicativo de configuración en proceso
- 4 entradas analógicas a través de bus I2C
- Protección hardware frene a cortocircuitos, polaridad inversa, etc.

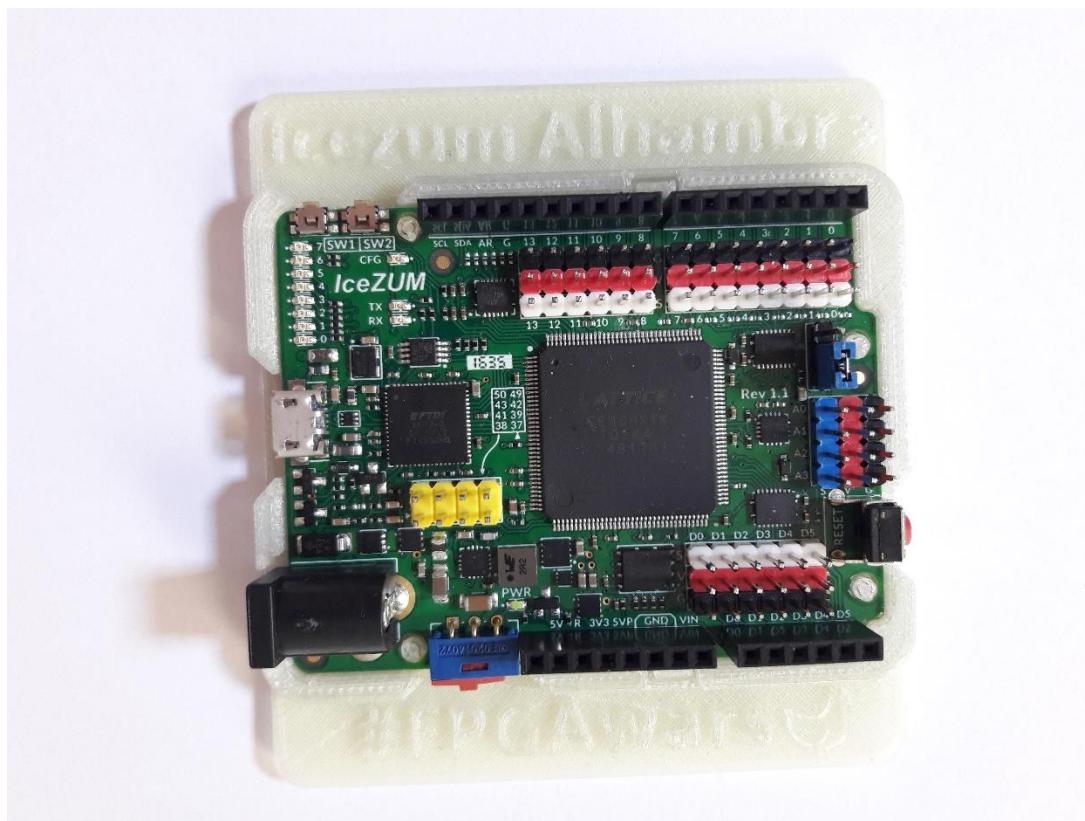


Figura 47. Placa IceZUM Alhambra

Existen muchas placas de desarrollo que contienen una FPGA y periféricos como la IceZUM Alhambra, como las de Xilinx, Altera o Papilio (algunas son libres y otras no) [4].

Ha sido diseñada por Eladio Delgado en colaboración con Juan González, con la idea de que pudiera, además, ser utilizada en educación. También ha colaborado Jesús Arroyo, aportando en herramientas como IceStudio.

De hecho, el punto más fuerte de esta placa es la comunidad que hay por detrás y a la que cada vez se van incorporando más personas. La comunidad dispone de muchos recursos, tales como tutoriales, repositorios, documentaciones detalladas y está presente en eventos, foros y redes sociales donde poder contactar tanto con otras personas de la comunidad como con los propios autores, muy activos y siempre dispuestos a ayudar. La comunidad se llama FPGA Wars. Incluso el propio Juan González, también conocido como Obijuan, ha creado una “Academia Jedi de Hardware” fomentando la participación y aprendizaje de la comunidad.



Figura 48. Academia Jedi de Hardware (Fuente: FPGA Wars)

### 3.3.2. Impresión de piezas 3D

La comunidad FPGA Wars pone a disposición de cualquier persona interesada un repositorio con piezas 3D que se pueden usar para ensamblar componentes electrónicos de una forma más compacta. En este caso, vamos a imprimir un “Alhambra-Switch”, una manera de acoplar interruptor, cables y conector en una sola pieza 3D y poder trabajar con mayor comodidad. [6]

Si es necesario modificar la pieza porque, por ejemplo, nuestro interruptor sea diferente al documentado en el repositorio, siempre podemos editar la pieza en algún programa CAD. Hay opciones con licencia libre o como en este caso, usar SketchUp de Google, siendo este último muy fácil de usar.

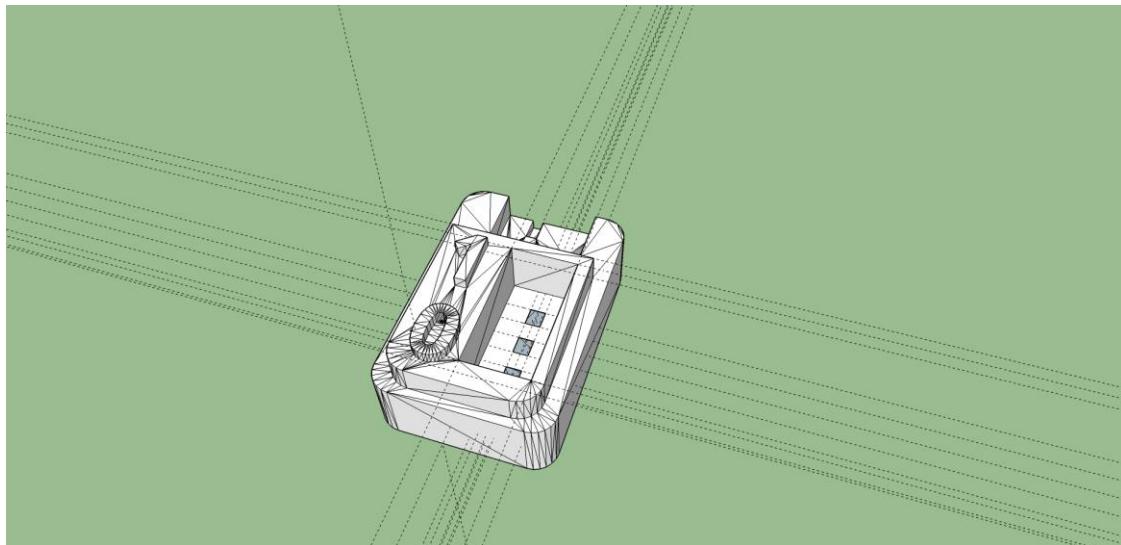


Figura 49. Modelo en 3D en SketchUp de Google listo para imprimir

Una vez que tienes tu modelo listo, solo tienes que exportarlo a STL o OBJ según tu impresora. Para ello, necesitamos un plugin que descargamos desde la Extension Warehouse de SketchUp. Cuando la tengamos instalada, estaremos listos para exportar nuestro modelo 3D a formato STL (o el requerido por la impresora) e imprimir.

### Soldadura de los interruptores a las piezas 3D con cables

*Nota: depende de las características del interruptor habrá que soldar de una forma o de otra. Por ejemplo, existen interruptores deslizantes que tienen seis patillas en lugar de tres. También hay que tener cuidado a la hora de conectar los pines del interruptor con los pines del conector correctos, ya que puede ocurrir que, por ejemplo, el interruptor funcione al revés (cuando debería estar a 1 está a 0 y cuando debería estar a 0 está a 1). Es muy recomendable analizar bien la hoja de especificaciones del componente en cuestión.*



Primero soldamos unos un cable pequeño a cada pin del interruptor (antes de esto, hemos tomado unas medidas aproximadas de cuánto cable es necesario en la pieza 3D)

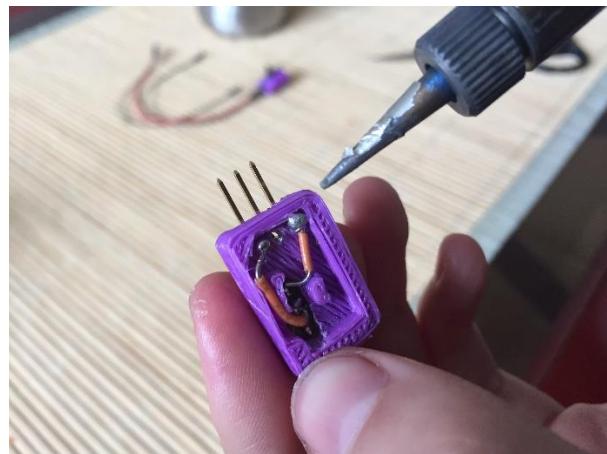


Colocamos los cables en sus posiciones para prepararlos para soldarlos



Así queda una vez soldado

Lo metemos en la pieza 3D en la posición adecuada (si no, puede que luego funcione al revés)



Soldamos



Y aquí tenemos nuestros ocho interruptores terminados

Figura 50. Proceso de soldadura de un Alhambra-Switch. Como resultado se muestran ocho de ellos.

Así quedarían los interruptores conectados ya a la placa:

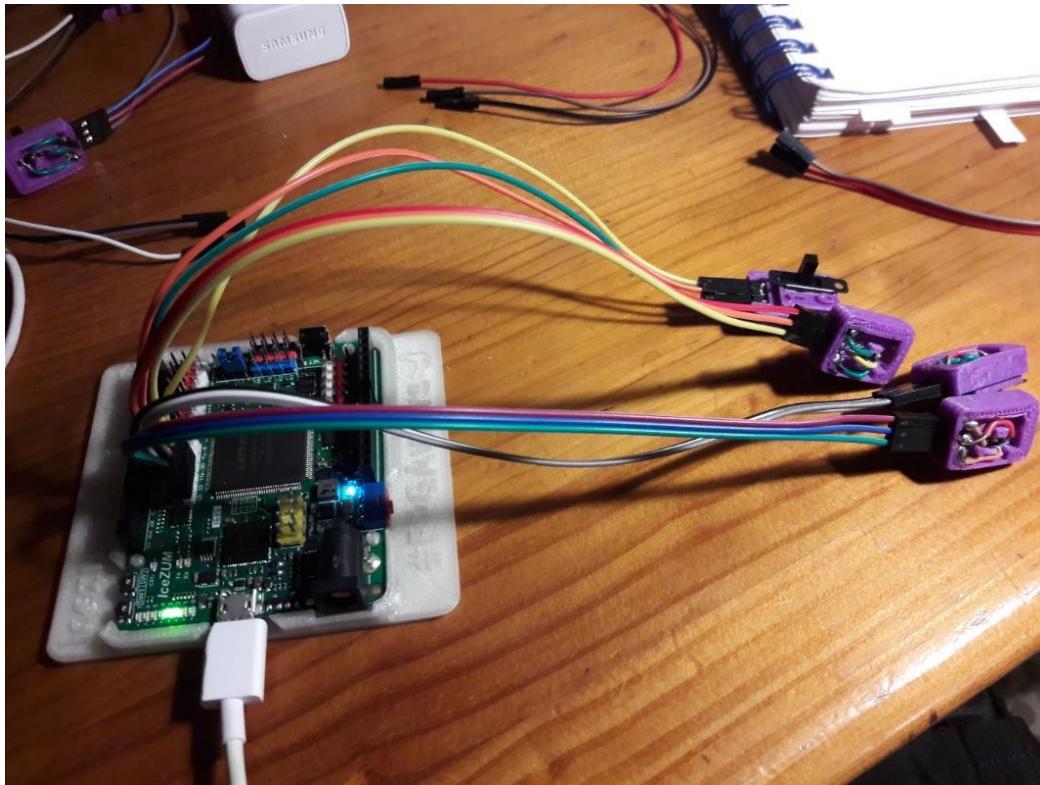


Figura 51. Interruptores conectados y un circuito (en concreto, un 4-Bit Right Rotate) cargado en la FPGA

### 3.3.3. PCB (Printed Circuit Board)

Otra opción para conectar interruptores y LEDs externos a la placa es usar placas de circuito impreso (PCB).

Una placa de circuito impreso es un componente electrónico hecho de una o más capas de material conductor que son separadas por un material aislante. Otros componentes electrónicos (tales como resistencias, transistores o LEDs) se colocan encima y/o debajo de la placa para crear un circuito eléctrico completo. El PCB tiene propiedades eléctricas, mecánicas y térmicas que deben ser tenidas en cuenta a la hora de crear un diseño para una aplicación concreta.

La herramienta libre que usaremos para diseñar nuestro PCB es Kicad. Cuando usemos este programa, pasaremos por varias fases. En la primera, definiremos los componentes que va a contener nuestro circuito y cómo se van a conectar entre s. Por ejemplo, cuatro LEDs y cuatro resistencias:

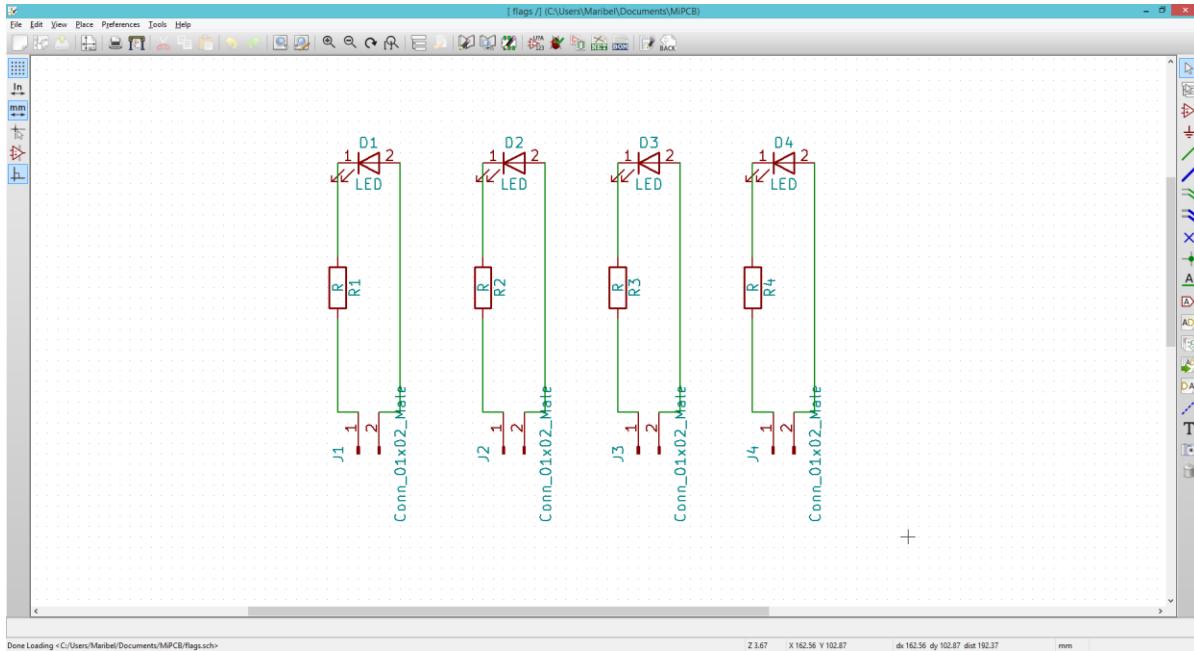
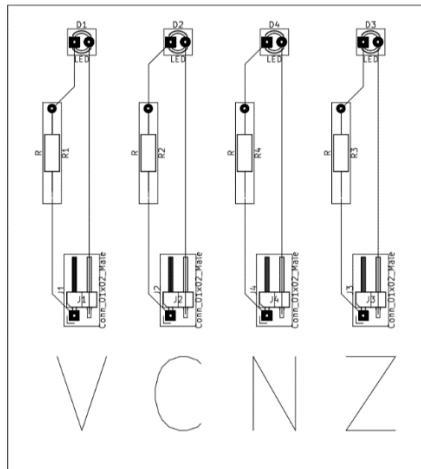


Figura 52. Diagrama de componentes en Kicad Primer paso a realizar dentro del programa.

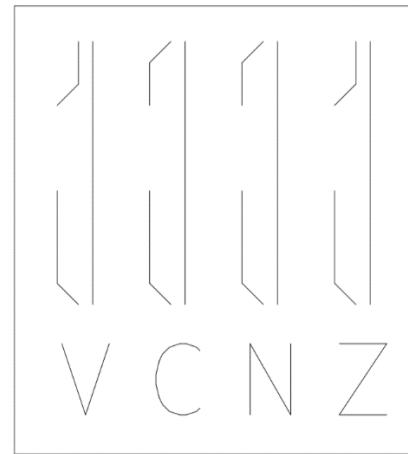
A continuación, generaremos una ***netlist***. Después, asociaremos manualmente un ***footprint*** a cada componente definido en la primera fase. Un footprint no es más que, gráficamente hablando, la vista de la planta de un componente. Esta vista nos servirá para saber cuánto espacio físico ocupará nuestro componente en la placa y nos ayudará a distribuir mejor los componentes en la superficie de esta.

Finalmente, llegamos a la fase de diseño, propiamente dicho, del PCB. En ella, distribuimos los elementos por toda la placa según nuestros requisitos. Seguidamente, podemos dejar que el programa “enrute”, es decir, trace las líneas de cobre entre componente y componente, automáticamente por nosotros, o podemos hacerlo a mano. También podemos usar capas para añadir referencias visuales como etiquetas y texto a nuestra placa.

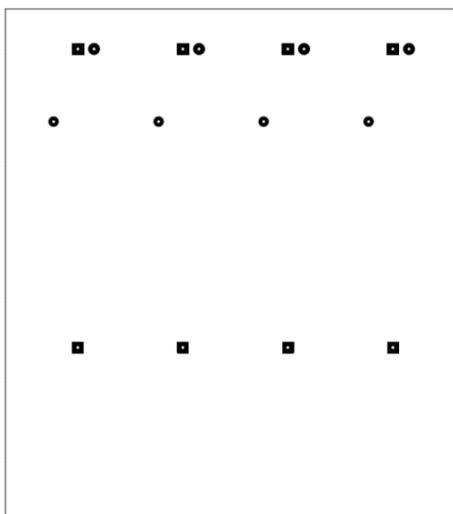
Como ejemplo, las diferentes capas de uno de los PCBs que vamos a fabricar son estas:



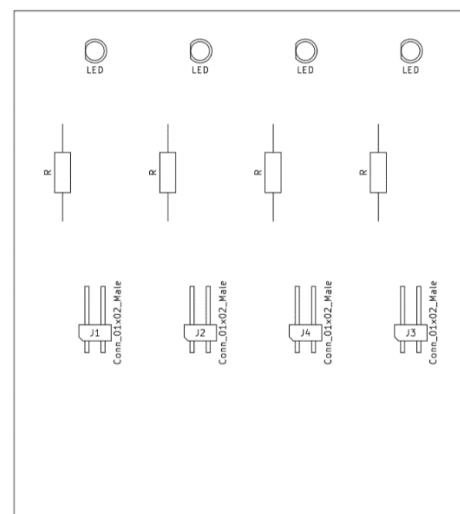
Todas las capas



Cobre del lado frontal



Pads del lado frontal



Referencias sobreimpresas del lado frontal

*Figura 53. Capas de un PCB.*

El diseño del otro PCB es el mostrado a continuación:

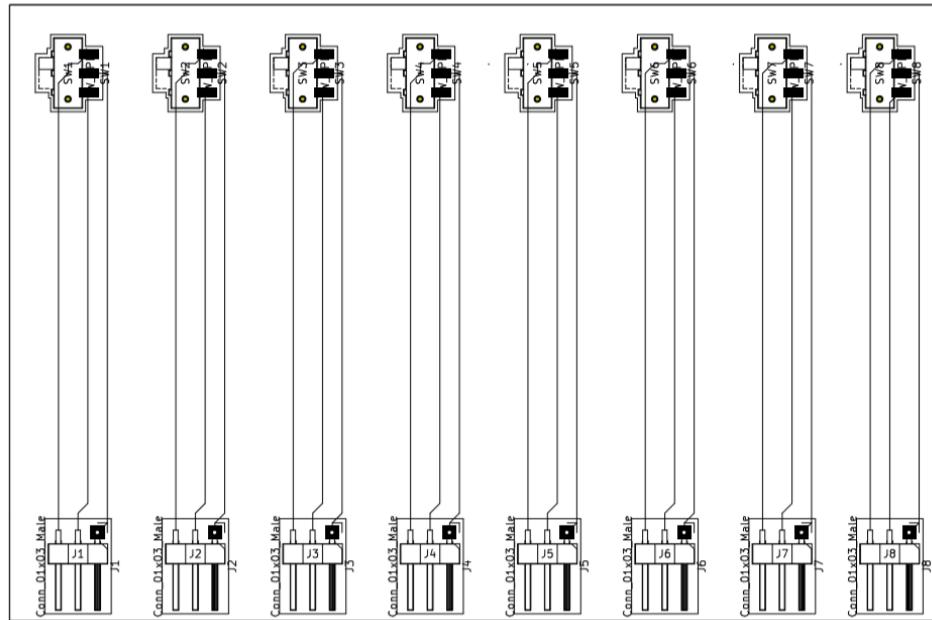


Figura 54. Todas las capas del segundo PCB

Solo nos queda, imprimir nuestro diseño, plasmarlo en la placa, grabarlo en ella y soldar los componentes [5]. Nuestras placas quedarían de la siguiente manera:

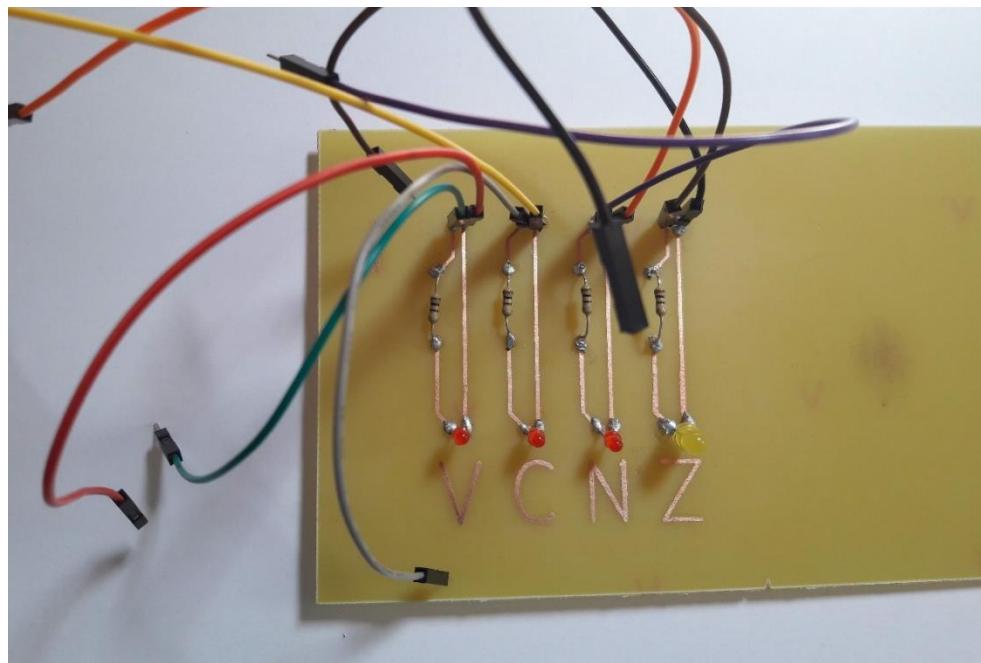


Figura 55. Contiene cuatro LEDs que representan los flags de la ALU

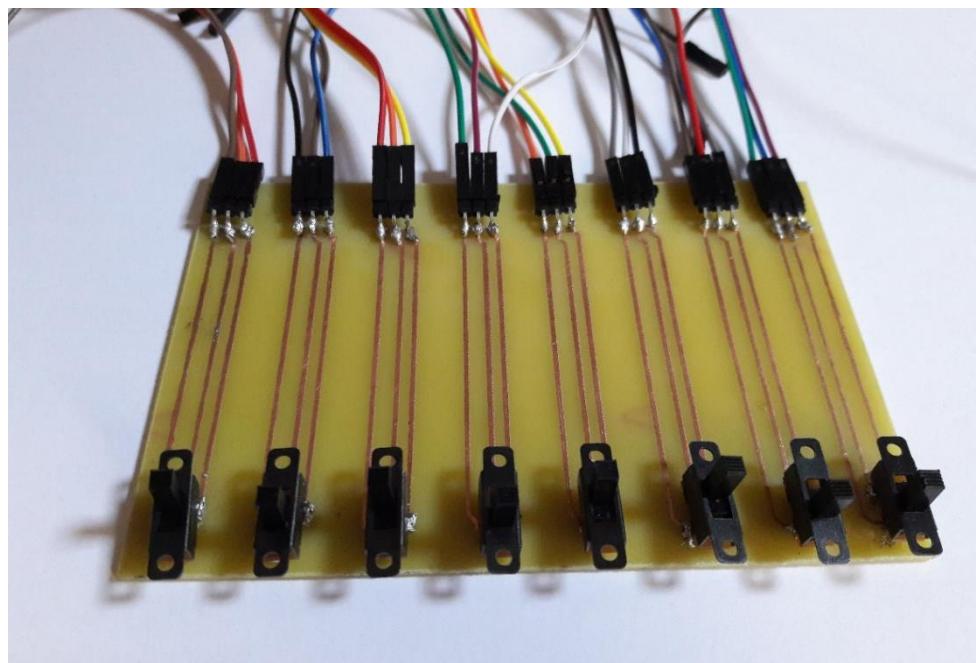


Figura 56. Contiene ocho interruptores: cuatro para la entrada A y cuatro para la entrada B, ambos de 4 bits

El montaje final se muestra a continuación

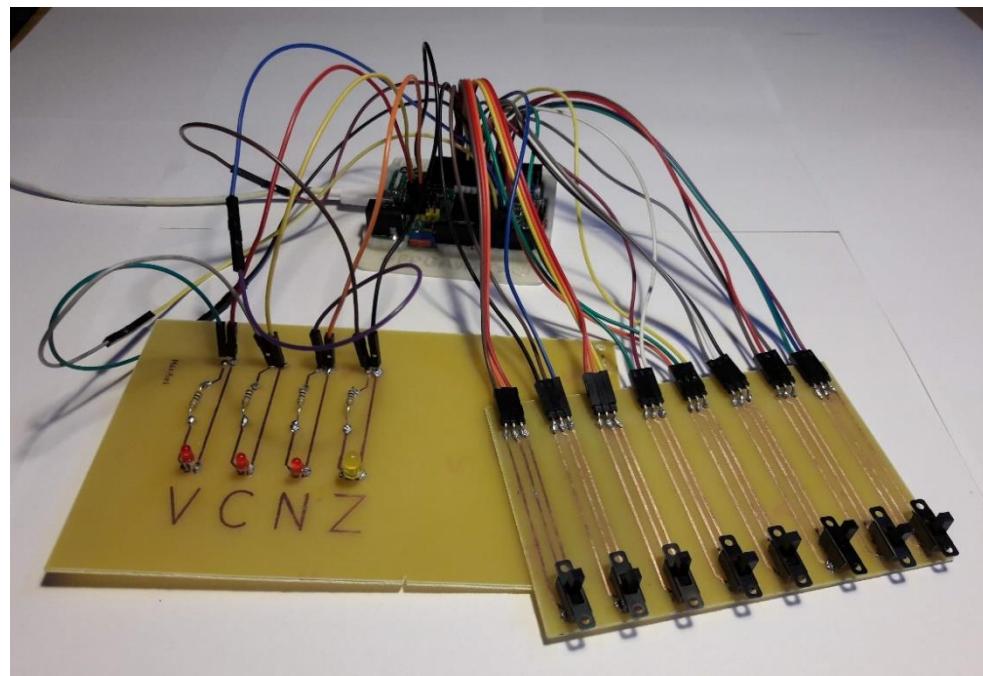


Figura 57. Montaje final

*Nota: En el Anexo se proveen varios esquemas para conectar los componentes a la IceZUM Alhambra.*

# 4. Conclusiones y trabajos futuros

Tras la realización de este trabajo hemos podido comprobar que las FPGAs son herramientas muy útiles para prototipar e implementar diseños digitales de forma eficiente. Nos ha servido para comprender por qué empresas importantes del sector tecnológico están apostando por esta tecnología. Debido a la gran cantidad de datos que se generan y deben ser procesados en los servidores de estas empresas, las FPGAs son una muy buena opción como acelerador del proceso de análisis frente a otras tecnologías, tales como GPUs o ASICS. Aunque su integración en los equipos actuales pueda resultar laboriosa (al fin y al cabo, se está incorporando un nuevo componente a todo un sistema en funcionamiento), los resultados lo valen: versatilidad ante cualquier cambio en la infraestructura del sistema actual, rapidez a la hora de efectuar esos cambios y eficiencia y optimización gracias al paralelismo masivo que ofrece su arquitectura hardware.

A juicio de la autora, el comienzo de la curva de aprendizaje en FPGAs puede ser elevado, principalmente, si no se tiene una base firme en electrónica. Sin embargo, el descubrimiento de divulgadores como Juan González, se ha podido confirmar que lo técnico no tiene por qué ser algo difícil de aprender, al contrario, si se utilizan las metodologías y recursos adecuados, se puede conseguir aprender de forma efectiva.

Gracias también al repaso e investigación previos en electrónica, la autora ha conseguido ampliar su visión global sobre la informática, especialmente, en el sentido de comprender cómo se complementan hardware y software entre sí, para crear los sistemas digitales cada vez más numerosos y complejos que se encuentran a nuestro alrededor.

Dos aspectos, en opinión de la autora, han quedado pendientes de mejorar. En el primero, los circuitos digitales mostrados en este proyecto carecen de optimización alguna, tanto la ALU como la lógica de control de esta. El objetivo principal ha sido implementar una ALU en concreto, un circuito digital en general, en una FPGA, aprovechando para conocer más a fondo esta tecnología. El segundo aspecto consiste en describir el hardware a implementar en la FPGA mediante HDL en lugar de hacerlo gráficamente. Usar código puede suponer una ventaja cuando son circuitos más grandes y/o complejos de lo normal, ya que aporta rapidez y flexibilidad en la tarea. Si el lector de este trabajo quisiera investigar y/o mejorar este proyecto, estos dos aspectos son dos vías sugeridas por la autora por las que comenzar.



# 5. Referencias

## Referenciadas en el texto (por orden de aparición)

- [1] BricoGeek. (2016, 7 noviembre). OSHWDem 2016: Charla FPGAs Libres - Juan González ( Obijuan ) [Archivo de vídeo ]. Recuperado de <https://www.youtube.com/watch?v=XWC1B7UKv98>
- [2] González, J. (s.f.). Tutorial de Electrónica Digital con FPGAs libres. Recuperado de <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial>
- [3] Charte, F., Espinilla, M., Rivera, A. J., & Pulgar, F. (2017). Uso de dispositivos FPGA como apoyo a la enseñanza de asignaturas de arquitectura de computadores. Recuperado de <http://sinbad2.ujaen.es/sites/default/files/publications/FPGA.pdf>
- [4] González, J. (s.f.). Vídeo 3: La Icezum Alhambra y otras placas con FPGAs libres. Recuperado 17 diciembre, 2017, de <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki/V%C3%ADdeo-3:-La-Icezum-Alhambra-y-otras-placas-con-FPGAs-libres>
- [5] DIY Customized Circuit Board (PCB Making). (2011, 3 septiembre). Recuperado de <http://www.instructables.com/id/Making-A-Customized-Circuit-Board-Made-Easy/>
- [6] González, J. (s.f.). Alhambra-switch. Recuperado 16 octubre, 2017, de <https://github.com/PCBPrints/Alhambra-switch/wiki>
- [7] Grout, I. (2008). *Digital systems design with FPGAs and CPLDs*. USA, USA: Newnes.
- [8] Petzold, C. (2000). *Code: The hidden language of computer hardware and software*. Redmond, USA: Microsoft Press.
- [9] Floyd, T. L. (2006). *Fundamentos de sistemas digitales*. Madrid, España: Pearson Educación S.A..
- [10] Wikipedia. (s.f.). Arithmetic logic unit. Recuperado 17 mayo, 2018, de [https://en.wikipedia.org/wiki/Arithmetic\\_logic\\_unit](https://en.wikipedia.org/wiki/Arithmetic_logic_unit)

- [11] Wikipedia. (s.f.). 74181. Recuperado 28 diciembre, 2017, de <https://en.wikipedia.org/wiki/74181>
- [12] Wikipedia. (s.f.). John von Neumann. Recuperado 28 mayo, 2018, de [https://es.wikipedia.org/wiki/John\\_von\\_Neumann](https://es.wikipedia.org/wiki/John_von_Neumann)
- [13] Wikipedia. (s.f.). Von Neumann architecture. Recuperado 20 mayo, 2018, de [https://en.wikipedia.org/wiki/Von\\_Neumann\\_architecture](https://en.wikipedia.org/wiki/Von_Neumann_architecture)
- [14] Signetics. (1985, 4 diciembre). 74181, LS181, S181 Arithmetic Logic Units. Recuperado de [http://pdf.datasheetcatalog.com/datasheets/560/493318\\_DS.pdf](http://pdf.datasheetcatalog.com/datasheets/560/493318_DS.pdf)
- [15] Ventura, V. (2014, 11 diciembre). ¿Qué es la lógica programada? Recuperado de <https://polaridad.es/logica-programable-que-es-pld-fpga-hdl-cpld/>
- [16] Wikipedia. (s.f.). JTAG. Recuperado 2 mayo, 2018, de <https://en.wikipedia.org/wiki/JTAG>
- [17] Microsoft. (s.f.). Project Catapult. Recuperado de <https://www.microsoft.com/en-us/research/project/project-catapult/>
- [18] Section 6.7 Shifts. (s.f.). Recuperado de <http://brahe.canisius.edu/~meyer/253/BOOK/ch6/FULLPAGE/ch6-7.html>

## **Consultadas (por orden de aparición)**

- Crespo, X. (2017, 18 enero). Qué es una FPGA y por qué jugarán un papel clave en el futuro. Recuperado de <https://planetachatbot.com/qu%C3%A9-es-una-fpga-y-por-qu%C3%A9-jugar%C3%A1n-un-papel-clave-en-el-futuro-e76667dbce3e>
- Johnson, J. (2011, 15 julio). List and comparison of FPGA companies. Recuperado de <http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html>
- Dani, S. (s.f.). IceZUM Alhambra. La FPGA libre que revolucionará la electrónica digital en educación.. Recuperado de <https://juegosrobotica.es/icezum-alhambra-educacion/>
- Rebolledo, M. A. (2010, 23 marzo). Diseño y elaboración de un texto para la programación en hardware basada en la plataforma de desarrollo FPGA Spartan 3A. Recuperado de [http://repository.upb.edu.co/bitstream/handle/20.500.11912/837/digital\\_19163.pdf?sequence=1&isAllowed=y](http://repository.upb.edu.co/bitstream/handle/20.500.11912/837/digital_19163.pdf?sequence=1&isAllowed=y)
- Macintosh Wire Wrap Logic Board #5 1980-1983. (s.f.). Recuperado de <http://www.digibarn.com/collections/partsmac-wirewrap5-board/>
- Trimberger, S. M. (2015, marzo). Three Ages of FPGAs: A Retrospective on the First Thirty Years of FPGA Technology. Recuperado de <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7086413>

- Scherr, T. (s.f.). 2. A Brief History of Programmable Logic. Recuperado de  
<https://es.coursera.org/learn/intro-fpga-design-embedded-systems/lecture/YaCfa/2-a-brief-history-of-programmable-logic>
- PCB Basics. (s.f.). Recuperado de <https://learn.sparkfun.com/tutorials/pcb-basics>
- Petzold, C. (s.f.). CODE The Hidden Language of Computer Hardware and Software by Charles Petzold -- Technical Addendum Version 0.90 --. Recuperado de  
<http://www.charlespetzold.com/code/CodeTechnicalAddendum.html>
- Digital Logic. (s.f.). Recuperado de <http://www.play-hookey.com/digital/>
- Motorola. (s.f.). 4-BIT ARITHMETIC LOGIC UNIT. Recuperado de  
<http://www.esi.uclm.es/www/isanchez/apuntes/ci/74181.pdf>
- Lattice Semiconductor. (2017, octubre). iCE40™ LP/HX Family Data Sheet. Recuperado de  
<https://www.mouser.es/datasheet/2/225/iCE40LPHXFamilyDataSheet-1022803.pdf>
- SPST, SPDT, and DPDT Switches Demystified. (s.f.). Recuperado de  
[http://musicfromouterspace.com/analogsynth\\_new/ELECTRONICS/pdf/switches\\_demystified\\_assembly.pdf](http://musicfromouterspace.com/analogsynth_new/ELECTRONICS/pdf/switches_demystified_assembly.pdf)
- Expovista TV. (2016, 18 agosto). IDF 2016: Intel CEO Brian Krzanich Day 2 Keynote [Archivo de vídeo ]. Recuperado de <https://www.youtube.com/watch?v=Psd2JKu0PSw>
- Amazon. (s.f.). Instancias F1 de Amazon EC2: Ejecute FPGA personalizadas en la nube de AWS. Recuperado de <https://aws.amazon.com/es/ec2/instance-types/f1/>
- Microsoft. (2014, 12 agosto). Large-Scale Reconfigurable Computing in a Microsoft Datacenter. Recuperado de <https://www.microsoft.com/en-us/research/wp-content/uploads/2014/06/HC26.12.520-Recon-Fabric-Pulnam-Microsoft-Catapult.pdf>
- Microsoft. (2017, 5 agosto). Inside the Microsoft FPGA-based configurable cloud. Recuperado de <https://azure.microsoft.com/es-es/resources/videos/build-2017-inside-the-microsoft-fpga-based-configurable-cloud/>



# **5. Anexo**

# Ejemplo de la datasheet de una ALU real de, en este caso, Signetics

**Signetics**

## 74181, LS181, S181 Arithmetic Logic Units

4-Bit Arithmetic Logic Unit  
Product Specification

### Logic Products

#### FEATURES

- Provides 16 arithmetic operations: ADD, SUBTRACT, COMPARE, DOUBLE, plus 12 other arithmetic operations
- Provides all 16 logic operations of two variables: Exclusive-OR, Compare, AND, NAND, NOR, OR, plus 10 other logic operations
- Full lookahead carry for high-speed arithmetic operation on long words

#### DESCRIPTION

The '181 is a 4-bit high-speed parallel Arithmetic Logic Unit (ALU). Controlled by the four Function Select inputs ( $S_0 - S_3$ ) and the Mode Control Input (M), it can perform all the 16 possible logic operations or 16 different arithmetic operations on active HIGH or active LOW operands. The Function Table lists these operations.

TYPE	TYPICAL PROPAGATION DELAY	TYPICAL SUPPLY CURRENT (TOTAL)
74181	22ns	91mA
74LS181	22ns	21mA
74S181	11ns	120mA

#### ORDERING CODE

PACKAGES	COMMERCIAL RANGE
Plastic DIP	$V_{CC} = 5V \pm 5\%$ ; $T_A = 0^\circ C$ to $+70^\circ C$ N74181N, N74LS181N, N74S181N

#### NOTE:

For information regarding devices processed to Military Specifications, see the Signetics Military Products Data Manual.

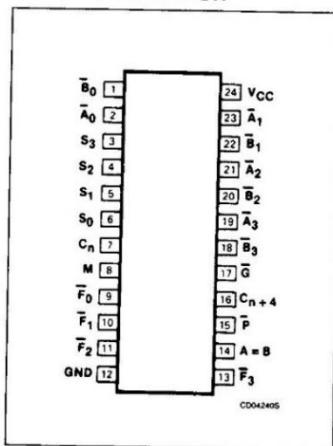
#### INPUT AND OUTPUT LOADING AND FAN-OUT TABLE

PINS	DESCRIPTION	74	74S	74LS
Mode	Input	1 <ul style="list-style-type: none">li</ul>	1 <ul style="list-style-type: none">Sul</ul>	1 <ul style="list-style-type: none">LSul</ul>
$\bar{A}$ or $\bar{B}$	Inputs	3 <ul style="list-style-type: none">li</ul>	3 <ul style="list-style-type: none">Sul</ul>	3 <ul style="list-style-type: none">LSul</ul>
S	Inputs	4 <ul style="list-style-type: none">li</ul>	4 <ul style="list-style-type: none">Sul</ul>	4 <ul style="list-style-type: none">LSul</ul>
Carry	Input	5 <ul style="list-style-type: none">li</ul>	5 <ul style="list-style-type: none">Sul</ul>	5 <ul style="list-style-type: none">LSul</ul>
$F_0 - F_3 = B, C_{n+4}$	Outputs	10 <ul style="list-style-type: none">li</ul>	10 <ul style="list-style-type: none">Sul</ul>	10 <ul style="list-style-type: none">LSul</ul>
$\bar{G}$	Output	10 <ul style="list-style-type: none">li</ul>	10 <ul style="list-style-type: none">Sul</ul>	40 <ul style="list-style-type: none">LSul</ul>
$\bar{P}$	Output	10 <ul style="list-style-type: none">li</ul>	10 <ul style="list-style-type: none">Sul</ul>	20 <ul style="list-style-type: none">LSul</ul>

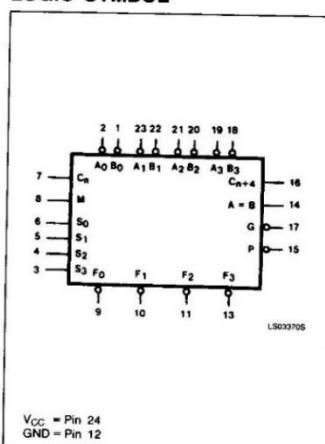
#### NOTE:

Where a 74 unit load (ul) is understood to be  $40\mu A I_{IH}$  and  $-1.6mA I_{IL}$ , a 74S unit load (Sul) is  $50\mu A I_{IH}$  and  $-2.0mA I_{IL}$ , and 74LS unit load (LSul) is  $20\mu A I_{IH}$  and  $-0.4mA I_{IL}$ .

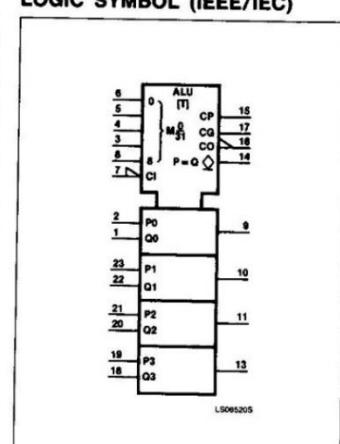
#### PIN CONFIGURATION



#### LOGIC SYMBOL



#### LOGIC SYMBOL (IEEE/IEC)



December 4, 1985

5-350

853-0540 81502

## Arithmetic Logic Units

## 74181, LS181, S181

When the Mode Control input (M) is HIGH, all internal carries are inhibited and the device performs logic operations on the individual bits as listed. When the Mode Control Input is LOW, the carries are enabled and the device performs arithmetic operations on the two 4-bit words. The device incorporates full internal carry lookahead and provides for either ripple carry between devices using the  $C_{n+4}$  output, or for carry lookahead between packages using the signals  $\bar{P}$  (Carry Propagate) and  $\bar{G}$  (Carry Generate).  $\bar{P}$  and  $\bar{G}$  are not affected by carry in. When speed requirements are not stringent, it can be used in a simple ripple carry mode by connecting the Carry output ( $C_{n+4}$ ) signal to the Carry input ( $C_n$ ) of the next unit. For high-speed operation the device is used in conjunction with the

'182 carry lookahead circuit. One carry lookahead package is required for each group of four '181 devices. Carry lookahead can be provided at various levels and offers high-speed capability over extremely long word lengths.

The  $A = B$  output from the device goes HIGH when all four  $\bar{F}$  outputs are HIGH and can be used to indicate logic equivalence over 4 bits when the unit is in the subtract mode. The  $A = B$  output is open collector and can be wired-AND with other  $A = B$  outputs to give a comparison for more than 4 bits. The  $A = B$  signal can also be used with the  $C_{n+4}$  signal to indicate  $A > B$  and  $A < B$ .

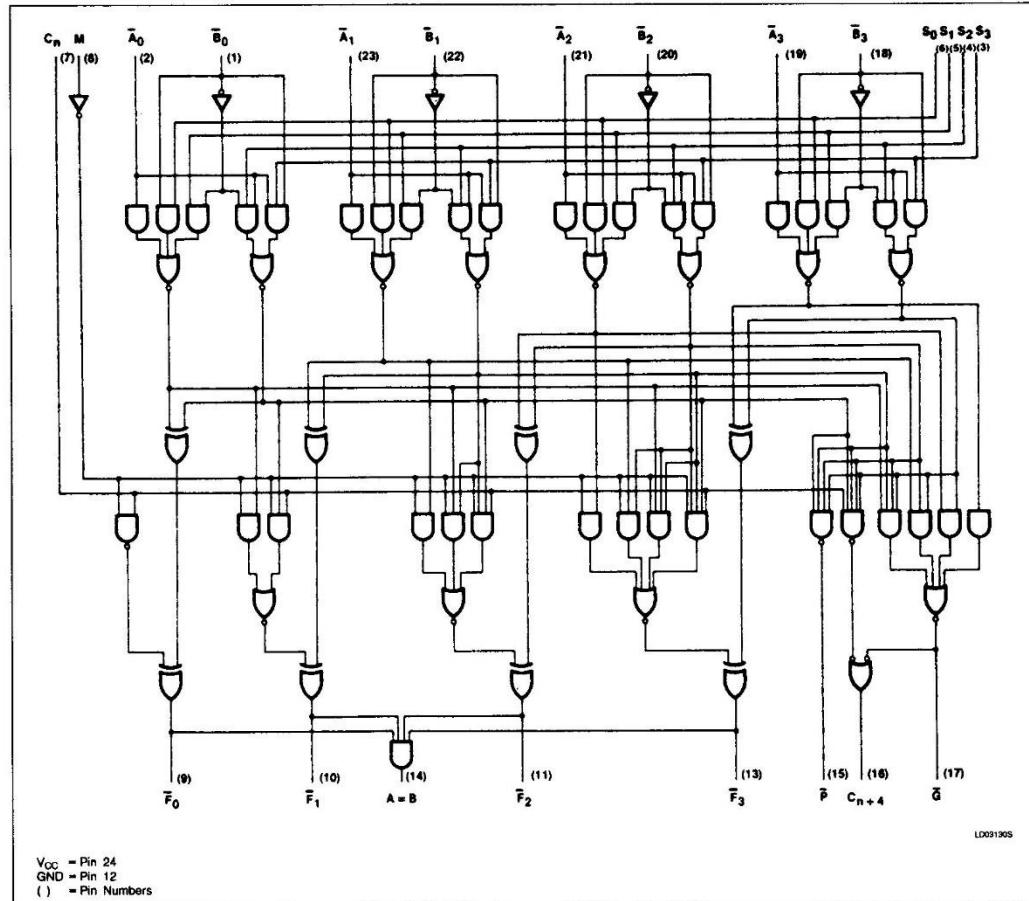
The Function Table lists the arithmetic operations that are performed without a carry in. An

incoming carry adds a one to each operation. Thus, select code LHHL generates A minus B minus 1 (2s complement notation) without a carry in and generates A minus B when a carry is applied.

Because subtraction is actually performed by complementary addition (1s complement), a carry out means borrow; thus, a carry is generated when there is no underflow and no carry is generated when there is underflow.

As indicated, this device can be used with either active LOW inputs producing active LOW outputs or with active HIGH inputs producing active HIGH outputs. For either case the table lists the operations that are performed to the operands labeled inside the logic symbol.

## LOGIC DIAGRAM



## Arithmetic Logic Units

74181, LS181, S181

MODE SELECT — FUNCTION TABLE

MODE SELECT INPUTS				ACTIVE HIGH INPUTS & OUTPUTS	
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Logic (M = H)	Arithmetic** (M = L) (C <sub>n</sub> = H)
L	L	L	L	$\bar{A}$	A
L	L	L	H	$\bar{A} + \bar{B}$	A + B
L	L	H	L	$\bar{A}\bar{B}$	A + $\bar{B}$
L	L	H	H	Logical 0	minus 1
L	H	L	L	AB	A plus AB
L	H	L	H	$\bar{B}$	(A + B) plus A $\bar{B}$
L	H	H	L	$A \bullet B$	A minus B minus 1
L	H	H	H	$\bar{A}\bar{B}$	AB minus 1
H	L	L	L	$\bar{A} + B$	A plus AB
H	L	L	H	$\bar{A} \bullet B$	A plus B
H	L	H	L	B	(A + $\bar{B}$ ) plus AB
H	L	H	H	AB	AB minus 1
H	H	L	L	Logical 1	A plus A*
H	H	L	H	$A + B$	(A + B) plus A
H	H	H	L	$A + B$	(A + $\bar{B}$ ) plus A
H	H	H	H	A	A minus 1

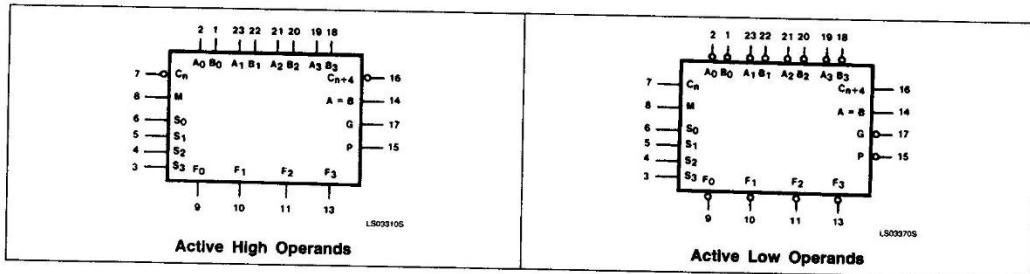
MODE SELECT INPUTS				ACTIVE LOW INPUTS & OUTPUTS	
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	Logic (M = H)	Arithmetic** (M = L) (C <sub>n</sub> = L)
L	L	L	L	$\bar{A}$	A minus 1
L	L	L	H	$\bar{A}\bar{B}$	AB minus 1
L	L	H	L	$\bar{A} + B$	$\bar{A}\bar{B}$ minus 1
L	L	H	H	Logical 1	minus 1
L	H	L	L	$\bar{A} + \bar{B}$	A plus (A + $\bar{B}$ )
L	H	L	H	$\bar{B}$	AB plus (A + $\bar{B}$ )
L	H	H	L	$\bar{A} \bullet B$	A minus B minus 1
L	H	H	H	$A + \bar{B}$	A + $\bar{B}$
H	L	L	L	$\bar{A}\bar{B}$	A plus (A + B)
H	L	L	H	$A \bullet \bar{B}$	A plus B
H	L	H	L	B	$\bar{A}\bar{B}$ (A + B)
H	L	H	H	$A + B$	A + B
H	H	L	L	Logical 0	A plus A*
H	H	L	H	$A\bar{B}$	AB plus A
H	H	H	L	AB	$\bar{A}\bar{B}$ plus A
H	H	H	H	A	A

L = LOW voltage

H = HIGH voltage level

\*Each bit is shifted to the next more significant position.

\*\*Arithmetic operations expressed in 2s complement notation.



## Arithmetic Logic Units

74181, LS181, S181

## ABSOLUTE MAXIMUM RATINGS (Over operating free-air temperature range unless otherwise noted.)

PARAMETER		74	74LS		74S		UNIT	
V <sub>CC</sub>	Supply voltage	7.0		7.0		7.0		V
V <sub>IN</sub>	Input voltage		-0.5 to +5.5		-0.5 to +5.5		-0.5 to +5.5	V
I <sub>IN</sub>	Input current		-30 to +5		-30 to +1		-30 to +5	mA
V <sub>OUT</sub>	Voltage applied to output in HIGH output state		-0.5 to +V <sub>CC</sub>		-0.5 to +V <sub>CC</sub>		-0.5 to +V <sub>CC</sub>	V
T <sub>A</sub>	Operating free-air temperature range				0 to 70			°C

## RECOMMENDED OPERATING CONDITIONS

PARAMETER	74			74LS			74S			UNIT	
	Min	Nom	Max	Min	Nom	Max	Min	Nom	Max		
V <sub>CC</sub>	Supply voltage	4.75	5.0	5.25	4.75	5.0	5.25	4.75	5.0	5.25	V
V <sub>IH</sub>	HIGH-level input voltage	2.0			2.0			2.0			V
V <sub>IL</sub>	LOW-level input voltage			+0.8			+0.8			+0.8	V
I <sub>IK</sub>	Input clamp current			-12			-18			-18	mA
I <sub>OH</sub>	HIGH-level output current			-800			-400			-1000	μA
I <sub>OL</sub>	LOW-level output current			16			8			20	mA
T <sub>A</sub>	Operating free-air temperature	0		70	0		70	0		70	°C

5

## SUM MODE TEST TABLE I

FUNCTION INPUTS: S<sub>0</sub> = S<sub>3</sub> = 4.5V, S<sub>1</sub> = S<sub>2</sub> = M = 0V

PARAMETER	INPUT UNDER TEST	OTHER INPUT, SAME BIT		OTHER DATA INPUTS		OUTPUT UNDER TEST
		Apply 4.5V	Apply GND	Apply 4.5V	Apply GND	
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	Ā <sub>i</sub>	None	Remaining Ā and Ā	C <sub>n</sub>	Ē <sub>i</sub>
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	Ā <sub>i</sub>	None	Remaining Ā and Ā	C <sub>n</sub>	Ē <sub>i</sub>
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	Ā <sub>i</sub>	None	None	Remaining Ā and Ā, C <sub>n</sub>	Ē
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	Ā <sub>i</sub>	None	None	Remaining Ā and Ā, C <sub>n</sub>	Ē
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	None	Ā <sub>i</sub>	Remaining Ā	Remaining Ā, C <sub>n</sub>	Ē
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	None	Ā <sub>i</sub>	Remaining Ā	Remaining Ā, C <sub>n</sub>	Ē
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	None	Ā <sub>i</sub>	Remaining Ā	Remaining Ā, C <sub>n</sub>	Ē
t <sub>PLH</sub> t <sub>PHL</sub>	Ā <sub>i</sub>	None	Ā <sub>i</sub>	Remaining Ā	Remaining Ā, C <sub>n</sub>	Ē
t <sub>PLH</sub> t <sub>PHL</sub>	C <sub>n</sub>	None	Ā <sub>i</sub>	All Ā	All Ā	Any Ē or C <sub>n+4</sub>

## Arithmetic Logic Units

74181, LS181, S181

DIFF MODE TEST TABLE II

FUNCTION INPUTS:  $S_0 = S_3 = 4.5V$ ,  $S_1 = S_2 = M = 0V$ 

PARAMETER	INPUT UNDER TEST	OTHER INPUT, SAME BIT		OTHER DATA INPUTS		OUTPUT UNDER TEST
		Apply 4.5V	Apply GND	Apply 4.5V	Apply GND	
$t_{PLH}$	$\bar{A}_i$	None	$\bar{B}_i$	Remaining $\bar{A}$	Remaining $\bar{B}, C_n$	$\bar{F}_i$
$t_{PHL}$	$\bar{B}_i$	$\bar{A}_i$	None	Remaining $\bar{A}$	Remaining $\bar{B}, C_n$	$\bar{F}_i$
$t_{PLH}$	$\bar{A}_i$	None	$\bar{B}_i$	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$\bar{P}$
$t_{PHL}$	$\bar{B}_i$	$\bar{A}_i$	None	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$\bar{P}$
$t_{PLH}$	$\bar{A}_i$	$\bar{B}_i$	None	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$\bar{G}$
$t_{PHL}$	$\bar{B}_i$	None	$\bar{A}_i$	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$\bar{G}$
$t_{PLH}$	$\bar{A}_i$	None	$\bar{B}_i$	Remaining $\bar{A}$	Remaining $\bar{B}, C_n$	$A = B$
$t_{PHL}$	$\bar{B}_i$	$\bar{A}_i$	None	Remaining $\bar{A}$	Remaining $\bar{B}, C_n$	$A = B$
$t_{PLH}$	$\bar{A}_i$	$\bar{B}_i$	None	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$C_{n+4}$
$t_{PHL}$	$\bar{B}_i$	None	$\bar{A}_i$	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$C_{n+4}$
$t_{PLH}$	$C_n$	None	None	All $\bar{A}$ and $\bar{B}$	None	Any $F$ or $C_{n+4}$

LOGIC MODE TEST TABLE III

PARAMETER	INPUT UNDER TEST	OTHER INPUT, SAME BIT		OTHER DATA INPUTS		OUTPUT UNDER TEST	FUNCTION INPUTS
		Apply 4.5V	Apply GND	Apply 4.5V	Apply GND		
$t_{PLH}$	$\bar{A}_i$	$\bar{B}_i$	None	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$\bar{F}_i$	$S_1 = S_2 = M = 4.5V$ $S_0 = S_3 = 0V$
$t_{PHL}$	$\bar{B}_i$	$\bar{A}_i$	None	None	Remaining $\bar{A}$ and $\bar{B}, C_n$	$\bar{F}_i$	$S_1 = S_2 = M = 4.5V$ $S_0 = S_3 = 0V$

## Arithmetic Logic Units

74181, LS181, S181

5

## DC ELECTRICAL CHARACTERISTICS (Over recommended operating free-air temperature range unless otherwise noted)

PARAMETER	TEST CONDITIONS <sup>1</sup>	74181			74LS181			74S181			UNIT	
		Min	Typ <sup>2</sup>	Max	Min	Typ <sup>2</sup>	Max	Min	Typ <sup>2</sup>	Max		
V <sub>OH</sub> HIGH-level output voltage	V <sub>CC</sub> = MIN, V <sub>IH</sub> = MIN, V <sub>IL</sub> = MAX, I <sub>OH</sub> = MAX	Any output except A = B	2.4	3.4		2.7	3.4		2.7	3.4		V
V <sub>OL</sub> LOW-level output voltage	V <sub>CC</sub> = MIN, V <sub>IH</sub> = MIN, V <sub>IL</sub> = MAX	I <sub>OL</sub> = MAX All outputs		0.2	0.4		0.35	0.5		0.5		V
		I <sub>OL</sub> = 4mA					0.25	0.4				V
		I <sub>OL</sub> = 16mA G output					0.47	0.7				V
		I <sub>OL</sub> = 8mA P output					0.35	0.5				V
V <sub>IK</sub> Input clamp voltage	V <sub>CC</sub> = MIN, I <sub>I</sub> = I <sub>IK</sub>				-1.5			-1.5			-1.2	V
I <sub>I</sub> Input current at maximum input voltage	V <sub>CC</sub> = MAX	Mode input			1.0			0.1			1.0	mA
		A or B inputs			1.0			0.3			1.0	mA
		S inputs			1.0			0.4			1.0	mA
		Carry input			1.0			0.5			1.0	mA
I <sub>IH</sub> HIGH-level input current	V <sub>CC</sub> = MAX	V <sub>I</sub> = 2.4V	Mode input		40							μA
			A or B inputs		120							μA
			S inputs		160							μA
			Carry input		200							μA
		V <sub>I</sub> = 2.7V	Mode input			20			50			μA
			A or B inputs			60			150			μA
			S inputs			80			200			μA
			Carry input			100			250			μA
I <sub>IL</sub> LOW-level input current	V <sub>CC</sub> = MAX	V <sub>I</sub> = 0.4V	Mode input		-1.6			-0.4				mA
			A or B inputs		-4.8			-1.2				mA
			S inputs		-6.4			-1.6				mA
			Carry input		-8			-2				mA
		V <sub>I</sub> = 0.5V	Mode input							-2		mA
			A or B inputs							-6		mA
			S inputs							-8		mA
			Carry input							-10		mA
I <sub>OH</sub> HIGH-level output current	V <sub>IH</sub> = MIN, V <sub>IL</sub> = MAX, V <sub>OH</sub> = 5.5V A = B only			250			100			250		μA
I <sub>OS</sub> Short-circuit output current <sup>3</sup>	V <sub>CC</sub> = MAX Any output except A = B	-18		-57	-15		-100	-40		-100		mA
I <sub>CC</sub> Supply current <sup>4</sup> (total)	V <sub>CC</sub> = MAX	Note 4a		88	140		20	34		120	220	mA
		Note 4b		94	150		21	37		120	220	mA

## NOTES:

- For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions for the applicable type.
- All typical values are at V<sub>CC</sub> = 5V, T<sub>A</sub> = 25°C.
- I<sub>OS</sub> is tested with V<sub>OUT</sub> = +0.5V and V<sub>CC</sub> = V<sub>CC</sub> MAX + 0.5V. Not more than one output should be shorted at a time and duration of the short circuit should not exceed one second.
- I<sub>CC</sub> is measured with the following conditions: a. S<sub>0</sub> through S<sub>3</sub>, M, and A inputs are at 4.5V, other inputs grounded, all outputs open. b. S<sub>0</sub> through S<sub>3</sub> and M inputs are at 4.5V, other inputs grounded, all outputs open.

## Arithmetic Logic Units

74181, LS181, S181

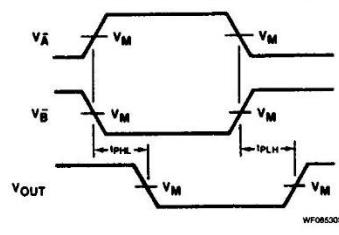
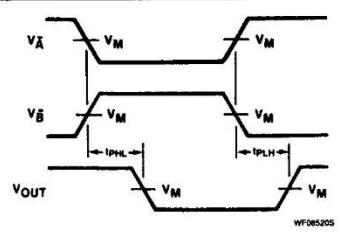
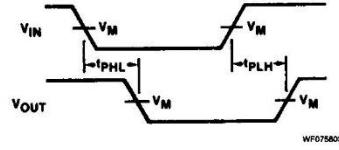
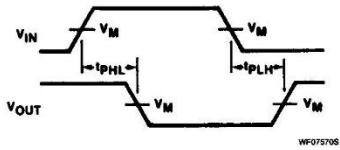
AC ELECTRICAL CHARACTERISTICS  $T_A = 25^\circ\text{C}$ ,  $V_{CC} = 5.0\text{V}$ 

PARAMETER	TEST CONDITIONS	74		74LS		74S		UNIT	
		$C_L = 15\text{pF}$ $R_L = 400\Omega$		$C_L = 15\text{pF}$ $R_L = 2\text{k}\Omega$		$C_L = 15\text{pF}$ $R_L = 280\Omega$			
		Min	Max	Min	Max	Min	Max		
$t_{PLH}$ $t_{PHL}$	Propagation delay $C_n$ to $C_{n+4}$	M = 0V, Sum or Diff Mode see Waveform 2 and Tables I & II		18 19		27 20		10.5 10.5 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $C_n$ to $\bar{F}$ outputs	M = 0V, Sum or Diff Mode see Waveform 2 and Tables I & II		19 18		26 20		12 12 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $\bar{B}$ inputs to $\bar{G}$ output	M = $S_1 = S_2 = 0\text{V}$ , $S_0 = S_3 = 4.5\text{V}$ Sum Mode, see Waveform 2 and Table I		19 19		29 23		12 12 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $B$ inputs to $\bar{G}$ output	M = $S_0 = S_3 = 0\text{V}$ , $S_1 = S_2 = 4.5\text{V}$ Diff Mode, see Waveform 3 and Table II		25 25		32 32		15 15 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $B$ inputs to $\bar{P}$ output	M = $S_1 = S_2 = 0\text{V}$ , $S_0 = S_3 = 4.5\text{V}$ Sum Mode, see Waveform 2 and Table I		19 25		30 30		12 12 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $B$ inputs to $\bar{P}$ output	M = $S_0 = S_3 = 0\text{V}$ , $S_1 = S_2 = 4.5\text{V}$ Diff Mode, see Waveform 3 and Table II		25 25		30 33		15 15 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}_i$ or $\bar{B}_i$ inputs to $\bar{F}_i$ outputs	M = $S_1 = S_2 = 0\text{V}$ , $S_0 = S_3 = 4.5\text{V}$ Sum Mode, see Waveform 2 and Table I		42 32		32 20		16.5 16.5 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}_i$ or $\bar{B}_i$ inputs to $\bar{F}_i$ outputs	M = $S_0 = S_3 = 0\text{V}$ , $S_1 = S_2 = 4.5\text{V}$ Diff Mode, see Waveform 3 and Table II		48 34		32 32		20 22 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}_i$ or $\bar{B}_i$ inputs to $\bar{F}_i$ outputs	M = 4.5V, Logic Mode see Waveform 2 and Table III		48 34		33 38		20 22 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $B$ inputs to $C_{n+4}$ output	M = 0V, $S_0 = S_3 = 4.5\text{V}$ , $S_1 = S_2 = 0\text{V}$ Sum Mode, see Waveform 1 and Table I		43 41		38 38		18.5 18.5 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $B$ inputs to $C_{n+4}$ outputs	M = 0V, $S_0 = S_3 = 0\text{V}$ , $S_1 = S_2 = 4.5\text{V}$ Diff Mode, see Waveform 4 and Table II		50 50		41 41		23 23 ns	
$t_{PLH}$ $t_{PHL}$	Propagation delay $\bar{A}$ or $B$ inputs to $A = B$ output	M = $S_0 = S_3 = 0\text{V}$ , $S_1 = S_2 = 4.5\text{V}$ Diff Mode, see Waveform 3 and Table II		50 48		50 62		23 30 ns	

## Arithmetic Logic Units

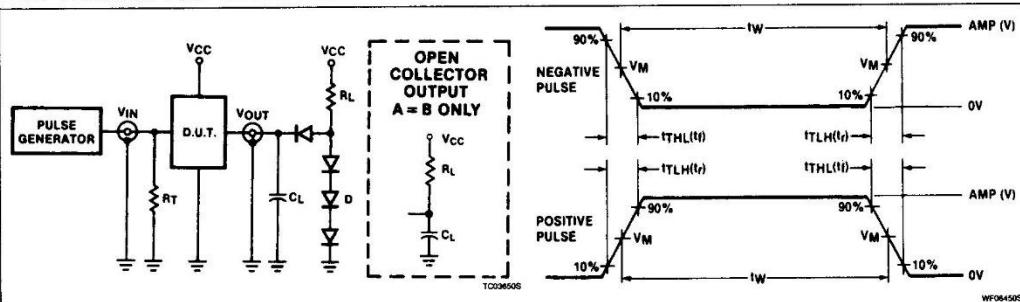
74181, LS181, S181

## AC WAVEFORMS



5

## TEST CIRCUITS AND WAVEFORMS



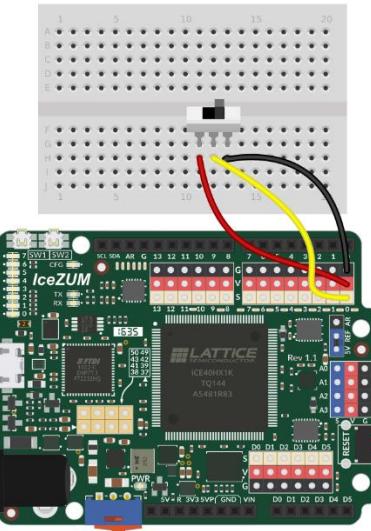
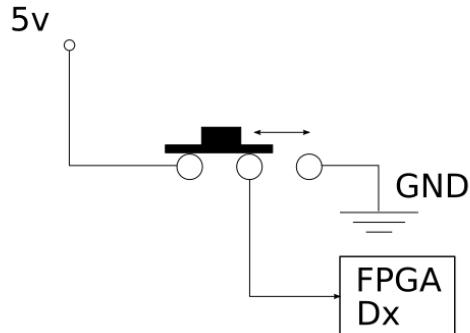
Test Circuit For 74 Totem-Pole Outputs

## DEFINITIONS

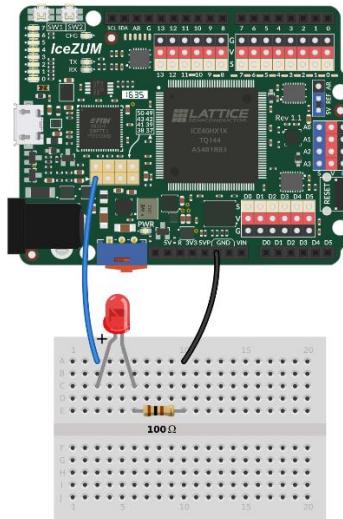
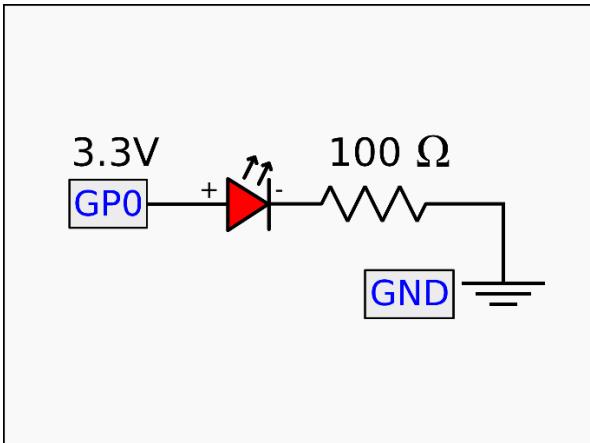
$R_L$  = Load resistor to  $V_{CC}$ ; see AC CHARACTERISTICS for value.  
 $C_L$  = Load capacitance includes jig and probe capacitance; see AC CHARACTERISTICS for value.  
 $R_T$  = Termination resistance should be equal to  $Z_{OUT}$  of Pulse Generators.  
 D = Diodes are 1N916, 1N3064, or equivalent.  
 $t_{TLH}, t_{THL}$  Values should be less than or equal to the table entries.

FAMILY	INPUT PULSE REQUIREMENTS				
	Amplitude	Rep. Rate	Pulse Width	$t_{TLH}$	$t_{THL}$
74	3.0V	1MHz	500ns	7ns	7ns
74LS	3.0V	1MHz	500ns	15ns	6ns
74S	3.0V	1MHz	500ns	2.5ns	2.5ns

## Esquemas de circuitos



Esquema para conectar un interruptor a la IceZUM Alhambra



Esquema para conectar un LED a la IceZUM Alhambra