Maribel

[Nombre de la compañía] | [Dirección de la compañía]

[Título del documento]

[Subtítulo del documento]

**ELECTRÓNICA DIGITAL DIVERTIDA CON FPGAS LIBRES**

Muchos podéis pensar, uff la Electrónica Digital es dificilísima, etc. Yo también pensaba eso hasta que vi una aplicación, que es un juego del móvil que os lo podéis instalar, que se llama “Circuit Scramble” y es Electrónica Digital y la gente está enganchada. Y es Electrónica Digital pura y dura: puertas AND, puertas OR, y mi sorpresa fue que a la gente le resulta fácil. Y dices, pues va a ser que la Electrónica Digital es intuitiva y ¡divertida! Lo que pasa que las herramientas que hemos tenido hasta ahora y la manera en la que nos han enseñado pues no han sido las más adecuadas, lo que no te motiva a seguir.

Entonces aquí vamos a hacer un ejemplo:

En IceStudio hacemos nuestro circuito digital y eso, mágicamente, se me va a mapear y se me mete dentro de la FPGA. Pero se mete físicamente. No hay emulación ni hay nada, no, no. El circuito que hagamos aquí en el programa físicamente se va a mapear aquí en la tarjeta y va a aparecer físicamente. Las FPGAs son como unos chips en blanco y lo que tienen ahí dentro son como todos los componentes electrónicos sólo falta pues establecer las uniones. Y eso es lo que hace que aparezcan nuestros circuitos.

En la práctica, pues es, hacemos nuestro circuito digital y nos aparece ahí. ¿Es como hacer software, no?, todo es súper rápido y muy fácil.

Aquí hay una demo.

Referencias:

Tutorial FPGAs Obijuan 1/x (vídeo);

<https://www.youtube.com/watch?v=R59Q-MwFbM8>

Tutorial FPGAs Obijuan 1/x (texto):

<https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki/Video-1:-Electr%C3%B3nica-digital-para-todos>

**Cómo funciona un microprocesador**

No sé a vosotros pero a mí me fascina la electrónica. No sé como se puede conseguir… todo esto a base, principalmente, de piedras y electricidad. Es posible que haya simplificado un poquito pero bueno, la esencia está ahí.

Hasta hace poco no lograba entender el funcionamiento de los aparatos electrónicos en general y ahora que lo sé estoy bastante a gusto la verdad.

Para entender este mundo de los aparatos electrónicos hay que tener en cuenta que el trabajo chulo, el pensar lo que es el cerebro es el microprocesador, lo que la gente suele llamar chip o microchip, los cosicas estas cuadradicas… eso. A grandes rasgos un microprocesador, conceptualmente, es un cacharro que tiene dos partes: una que decide lo que hacer y otra que lo hace, llamadas sorprendentemente (ohhh!) Unidad de Control y Unidad de Cálculo. La UCo lee una instrucción que forma parte de un programa, decide lo que hay que hacer y la UCa lo hace. Un programa de ordenador, por ejemplo, es una serie de instrucciones de este tipo. El microprocesador (nos referimos al entero) las lee y hace lo que le piden. Así de fácil! Hala, ya os podéis ir. XD

Nooo… es broma… No me voy a meter en el ordenador porque eso es un poco complejo y es otro mundo (hmmm…. O no…). Pero además tampoco hace falta saber cómo funciona el Intel core 7 duo 24 para entender cómo funciona la mayoría de aparatos electrónicos.

La mayoría de chips que hay en los electrodomésticos son cacharricos muy simples que utilizan tecnología de hace unos treinta años pero aún así son lo más, no os creáis. Estos realizan unas operaciones muy sencillas: mover datos, sumarlos, restarlos y si tienes pues, muchísima suerta, hasta los multiplica! Pero no os creáis que con esto se puede hacer casi cualquier cosa! Puedes hacer una lavadora, un USB, un MP3 o robots! Sí, sí, sí, robots, con lo que molan!

Para poder meternos un poco más en harina vamos a ver lo básico de la electrónica digital. Tenemos un circuitillo que se dedica a hacer operaciones. Este circuito tiene unas entradas, realiza la operación que haga sobre esas entradas y al ratito o al ratazo (que lo mismo se tira 200 nanosegundazos!), pues a ese rato saca unas salidas. Las operaciones que hacen son operaciones binarias, que básicamente coges ceros y unos, que haces operaciones muy sencillitas sobre ellos, y devuelves otros ceros y unos que pueden ser, yo qué sé: señales para encender un LED o esta pantalla, ahora mismo (a lo mejor es LED!). Si las entradas y las salidas de estos circuitos se guardan obtenemos LA MAGIA (seriedad). Porque si no sería un caos: al cambiar las entradas cambiarían las salidas cada una a su bola, y eso no puede ser, NO PUEDEN IR LAS SALIDAS CADA UNA A SU BOLA. Y además permiten la realimentación que es llevar alguna de las salidas como entrada y así poder tomar decisiones en función a los datos que se han generado.

Si volvemos al microprocesador, la unidad de proceso o cálculo de la que hablamos es muy fácil de explicar: tiene un circuito que realiza una operación, a ese circuito le llegan unas entradas y él saca unas salidas. Puede realizar varias operaciones y la que realice en cada momento dependerá de unas señales que vienen desde la unidad de control. La unidad de control tiene un poco más de chicha pero tampoco demasiada: principalmente es una memoria que contiene una serie de instrucciones que a su vez contienen la información sobre las operaciones que se han de realizar y sus operandos. Las instrucciones están en una lista, se lee una, se hace lo que se dice y se lee la siguiente. A menos que la instrucción indique que haya que realizar un salto. En ese caso se leerá la instrucción de la posición indicada.

Lo que marca el ritmo de cuándo se lee cada instrucción es una señal llamada señal de reloj, sorprendentemente también (ohhh!). Normalmente cuando pasa de 0 a 1 se renuevan los registros.

Por lo tanto, el funcionamiento general es el siguiente: el reloj pasa de 0 a 1, se lee una instrucción, esta instrucción contiene codificada con ceros y unos la operación que hay que realizar y los operandos o en su defecto dónde están los operandos. La parte que indica la operación se descodifica y se mandan unas señales de control a la unidad de proceso para que esta sepa que operación tiene que realizar. La parte de datos hace que se lleve como entradas a la unidad de proceso los operandos que venían indicados en la instrucción o el contenido de los registros que venían indicados en la instrucción. Además, especifica dónde se lleva el resultado generado. De esta manera, el procesador puede leer datos, realizar operaciones sobre ellos y, además, tomar decisiones según los resultados. Puesto que se pueden realizar saltos según los resultados obtenidos en las operaciones debido a la realimentación de la que antes hablábamos. Y ya está (como elvisayomastercard).

Espero que todo esto os haya parecido interesante, que hayáis aprendido algo.

Referencia:

Cómo funciona un microprocesador:

<https://www.youtube.com/watch?v=YKvTbp6ynsk>

**El transistor explicado para no electrónicos**

Hace breve repaso de la historia del transistor, su diferencia con los de germanio, qué es lo que significa la palabra y por qué, ejemplo interactivo.

Fuente: <https://www.youtube.com/watch?v=558eFxgz1Dc>

**¿Qué es un transistor?**

Muy buen símil con un grifo de agua, sobretodo después de haber visto el vídeo de los de germanio interactivo.

Fuentes: <https://www.youtube.com/watch?v=AYy-YejQdqA>

<http://electrico.scienceontheweb.net/resistencia4.html>

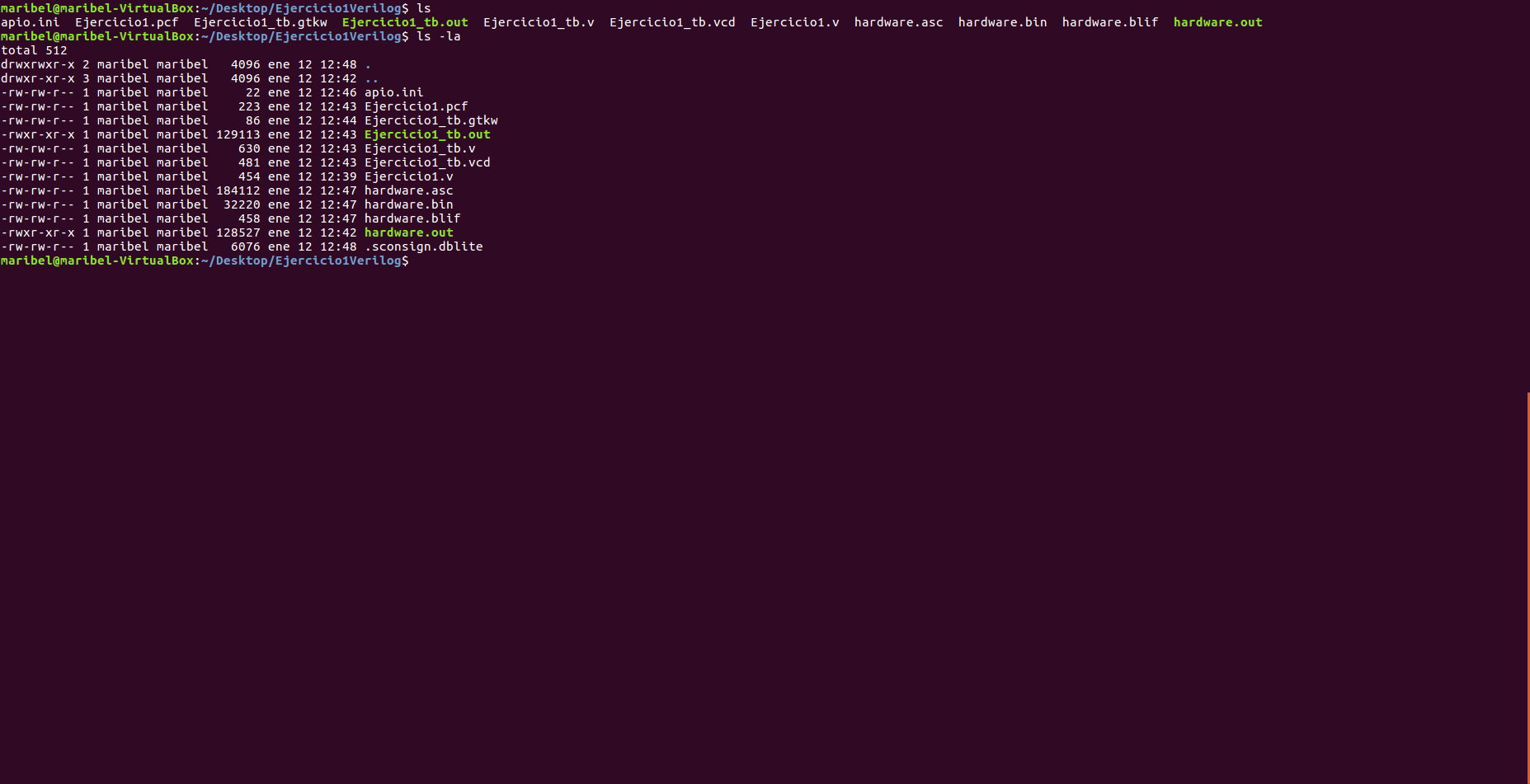
Esto es TOP: <https://www.allaboutcircuits.com/textbook/digital/chpt-16/microprocessor-programming/>

*Transistores – Qué son – Para qué sirven - Qué tienen por dentro – Silicio – Dopaje - Electricidad – Átomos – Electrones – Valencia*

VERILOG

Para probar Verilog, utilizamos una herramienta llamada Apio. La instalamos según las instrucciones que vienen en su documentación.

Partimos de un proyecto en IceStudio. Una vez lo tengamos terminado (por ahora he probado habiéndolo hecho visualmente, sin tocar código Verilog dentro del editor), tenemos la opción de exportarlo a diferentes formatos. En nuestro caso, necesitamos cuatro para poder simularlo en gtkwave, un simulador: un archivo verilog, un testbench, un pcf y un gtkwave (\*\*\* ver para qué sirve cada uno). Tenemos que crear también un archivo init que indique para qué placa estamos construyendo, en este caso la icezum. Una vez tengamos todos los archivos, podemos simular o ejecutarlo en la placa. Importante que el nombre de los archivos no contenga espacios o puntos porque puede dar errores.



En esta imagen se ven los archivos necesarios junto con los que se han generado al simular y ejecutar en la placa.

Por ahora, tanto en .v como en \_tb.v y .pcf hay que poner las salidas y el cable en mayúscula. Tras distintas pruebas, hay algunas que sí funcionan y otras que no, por eso apuesto por lo seguro.

Ahora, quiero seguir los tutoriales de Verilog para así después, probar a meter una entrada analógica y después probar a implementar un filtro.

Referencias:

<https://github.com/Obijuan/open-fpga-verilog-tutorial/wiki/Cap%C3%ADtulo-1%3A-%C2%A1Hola-mundo%21-Setbit>

<http://apiodoc.readthedocs.io/en/stable/index.html>

<http://gtkwave.sourceforge.net/gtkwave.pdf>

Para hacer los conectores Switch para la Icezum Alahambra (<https://github.com/PCBPrints/Alhambra-switch/wiki>) podemos hacer dos cosas:

* Cables hembra-hembra, tira de 3 conectores acodados (<https://www.electronicaembajadores.com/es/Productos/Detalle/CTO1MA18/conectores/conectores-regletas-de-pines/regleta-pin-2-54mm-macho-acodado-18-contactos>) y la impresión 3D. Si es necesario cortar la tira de tres en tres, se puede hacer con golpes secos con un cúter en la zona por donde se quiera partir.
* Conectores no polarizados de 3 pines, terminal hembra para conector poste no polarizado (<https://www.electronicaembajadores.com/es/Productos/Detalle/CTK5TTH/conectores/conectores-poste-no-polarizado/terminal-hembra-para-conector-poste-no-polarizado-2-54mm>) que conectará internamente los cables y el conector acodado, tira de 3 conectores acodados y la impresión 3D.

**ÍNDICE**

1. **Introducción**
   1. **Motivación y contexto**
   2. **Objetivos**
   3. **Fases de desarrollo (metodología) (esfuerzo en cada parte)**
2. **Estado del arte (de los más general a lo más específico del proyecto, no separar en bloques)**
   1. **Magnitudes analógicas y digitales**
   2. **ALU**
      1. **Arquitectura**
         1. **Arquitectura de Von Neumann**
      2. **Elementos (\*\*\* estaría dentro de arquitectura?)**
      3. **Historia (\*\*\* poner antes o después? Saldrá Von Neumann?)**
      4. **Datasheet**
      5. **Operaciones**
         1. **Funciones aritméticas (\*\*\* ver sección Desarrollo para más detalles)**
         2. **Funciones lógicas (\*\*\* ver sección Desarrollo para más detalles)**
   3. **Lógica programable**
      1. **Concepto básico de matriz AND**
      2. **Tecnologías de proceso basadas en conexiones programables**
         1. **Fusible**
         2. **Antifusible**
         3. **EPROM**
         4. **EEPROM**
         5. **SRAM**
      3. **Tipos de dispositivos lógicos programables (PLD)**
         1. **SPLD**
            1. **PAL**
            2. **GAL**
            3. **Macroceldas**
         2. **CPLD**
            1. **PLA(\*\*\* ?)**
         3. **FPGA**
            1. **Bloques lógicos configurables (CLB)**
            2. **LAB (Altera) (\*\*\* ?)**
            3. **IceZUM Alhambra**

**Características**

**Flujo de trabajo**

* + - 1. **Proceso de configuración (\*\*\* describirlo para los PLD en general o para MI FPGA?)**
         1. **Introducción del diseño**

**Lenguajes de descripción de hardware (HDL)**

**Verilog**

**VHDL**

**Esquemático (el que usamos nosotros)**

* + - * 1. **Simulación funcional**
        2. **Síntesis**
        3. **Implementación**
        4. **Simulación de temporización**
        5. **Programación del dispositivo**
        6. **Analogía urbanística**
      1. **JTAG**

1. **Desarrollos**
   * 1. **Funciones aritméticas de 4 bits**
        1. **Suma**
        2. **Resta**
        3. **Incrementador**
        4. **Decrementador**
     2. **Funciones lógicas de 4 bits**
        1. **Desplazamiento a izquierda y derecha**
        2. **Rotación a izquierda y derecha**
        3. **AND**
        4. **OR**
        5. **XOR**
        6. **Complemento a uno**
     3. **Circuitos útiles**
        1. **Multiplexor de 4 bits**
     4. **Impresión 3D**
     5. **PCB**
2. **Conclusiones y trabajos futuros**

* **Estado del arte: electrónica digital. Electrónica programable (Verilog, VHDL…). FPGA. Mi FPGA (lenguaje gráfico, características técnicas, capacidad… cómo se trabaja con ella)**

**CPU y microprocesadores (arquitectura von Neuman…, muy breve porque es genérico, mientras más a la izquierda más breve). ALU (arquitectura, partes, hablar de alguna real, poner en apéndice un ejemplo de datasheet de una ALU, justificar las operaciones: por historia, artículo de von Neunam, cómo describir los circuitos (coger ideas de las datasheets)).**

* **Desarrollos. Mi ALU, mis circuitos, el lenguaje (las combinaciones de los interruptores son instrucciones). Cuando cuente lo que hace cada instrucción, poner “reservado para uso futuro”. Proceso de desarrollo.**

**Explicar por qué he tomado X decisiones (ventajas, inconvenientes…).**

**2 x Interruptor + 2 x puerta triestado por bit de A.**

**Registro A, registro B, registro de salida. Para guardar la información.**

**Primero pongo la Address en el multiplexor. Poner un interruptor haces que se carguen los registros y se ejecute la operación. Al principio, poner otro interruptor que mande el paso del resultado al A de nuevo. Para hacerlo automáticamente, meto un contador. Al inicio se pone a 0. Si cuenta hasta 4, activa la grabación de la salida.**

**Introducción**, 8

**Motivación**, 8

**Objetivo**, 9

**Fases de desarrollo**, 9

**Estado del arte**, 10

**Circuitos Electrónicos: Analógicos y Digitales**, 10

**Continuo versus Discreto**, 10

**Analógico versus Digital**, 11

**Historia de la Lógica Digital**, 11

**Lógica programable versus Lógica Discreta**, 12

**Lógica programable versus procesadores**, 12

**Tipos de Lógica Programable**, 13

**Simple Programmable Logic Device**, 13

**Complex Programmable Logic Device**, 14

**Field Programmable Gate Array (FPGA)**, 14

**Fabricantes de Lógica Programable**, 15

**Metodologias y herramientas de diseño de lógica programable**, 15

**Introducción**, 15

**Flujo de diseño típico de un PLD**, 16

**Etapa 1: diseño de la ALU en Logisim (simulador)**, 28

**Etapa 2: trasnferencia del diseño en Logisim a IceStudio para su implementación en la FPGA**, 39

**Etapa 3: pruebas en la FPGA**, 40

**Conclusiones**, 45

**Glosario**, 45

**Referencias**, 46

**Detalles del proceso**, 28

1. **Introducción**
   1. **Motivación**

Desde que comencé el grado hasta que lo he acabado, he sentido curiosidad por saber cómo funcionaba un ordenador por dentro. A pesar de que me gusta el desarrollo de software, siento la necesidad de conocer el interior de esa máquina que programo. Dada la naturaleza del grado, no he tenido muchas asignaturas sobre hardware, por lo que he querido aprovechar este trabajo para profundizar en conceptos de más bajo nivel, y más ahora que, habiendo aprendido conceptos nuevos sobre alto nivel (software) quiero conectar estas dos partes tan importantes e interesantes que conforman los ordenadores de hoy en día: hardware y software. Para ello, vamos a usar un dispositivo lógico programable o PLD, en concreto, una FPGA, en la cual implementaremos una Unidad Aritmético Lógica (ALU) con el uso de herramientas libres.

* 1. **Objetivo**

Investigar y conocer más a fondo la relación entre software y hardware. Para llevar a cabo este trabajo, he necesitado repasar conceptos de electrónica digital, partiendo de los componentes básicos hasta llegar a cómo combinarlos para formar circuitos complejos. Como resultado práctico, diseñaremos el circuito electrónico de una ALU y lo implementaremos en una FPGA.

* 1. **Fases de desarrollo (visión global)**

Mi objetivo original era bajar de nivel tanto como fuera posible. Se barajó llegar hasta electrónica analógica (transistores…) (\*\*\* no sé si la rama se llama así) pero se decidió que era un tema fuera de mi alcance, porque no he cursado asignaturas que traten sobre ello, no tenía conocimientos ni experiencia propios suficientes como para comenzar por ahí y, además, podía quedarse fuera de un trabajo final relacionado con mi carrera.

Así que mi tutor me sugirió investigar sobre las FPGAs. Encontré un vídeo de Juan González, doctor en Robótica por la Universidad Autonoma de Madrid y actualmente profesor e investigador en la Univeridad Rey Juan Carlos de Madrid, también conocido como “Obijuan” en la comunidad, en la que explicaba de forma muy sencilla y amena qué eran las FPGAs, para qué servían y cómo se usaban [0]. Esto despertó aún más mi curiosidad sobre el tema. Así que decidí meterme en la comunidad y descubrí que tenían una tarjeta de desarrollo con una FPGA integrada llamada IceZUM Alhambra. Tras ver todo el movimiento y actividad que había, decidí comprar una y trabajar con ella. Seguí los tutoriales de Obijuan para ir aprendiendo.

Tras decidir que iba a usar una FPGA, se planteó hacer un filtro digital, pero pasaba el tiempo y mi falta de conocimientos y experiencia hacía que no consiguiera ni si quiera diseñar un boceto del hardware. Así que se optó por hacer algo más relacionado con lo que había estudiado en mi grado 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 para mí era 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, mi propio lenguaje de programación a muy bajo nivel.

Como preparación, leí un libro, recomendado por un profesor, llamado “Code”, el cual explica de manera sencilla y amena 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 [21]. Además de esta lectura a nivel teórico, investigué más a fondo sobre dispositivos lógicos programables (PLDs), de los cuales un tipo es la FPGA y comencé a realizar ejemplos cada vez más complejos sobre mi placa, para ir practicando. Esta parte fue la más larga y, para mí, más difícil, dado que tenía pocos conocimientos sobre hardware y electrónica en general.

Fijada la tecnología que iba a usar (la FPGA) y habiendo comenzado el diseño de la ALU, comencé a descargar e imprimir piezas 3D desde la plataforma de la comunidad para formar interruptores que pudiera conectar a la FPGA que me sirvieran como entrada de datos a la ALU.

1. **Estado del arte**
   1. **Magnitudes analógicas y digitales**

Antes de ver en detalle qué es un PLD y cómo usarlo, es importante identificar la naturaleza digital de los PLDs. Aquí veremos de forma global las características principales y las diferencias entre los mundos digital y analógico.

Hay dos tipos de circuitos electrónicos: digitales y analógicos. La electrónica digital utiliza magnitudes con valores discretos y la electrónica analógica emplea magnitudes con valores continuos.

Una magnitud analógica es aquella que toma valores continuos. Una magnitud digital es aquella que toma valores discretos. La mayoría de los elementos que podemos medir cuantitativamente en la naturaleza aparecen en forma analógica. Por ejemplo, a lo largo del día, la temperatura no varía instantáneamente de 20º a 25º instantáneamente, hay todo un rango de valores intermedios infinitos por los que pasa hasta llegar a 25º. Si dibujáramos una gráfica de esta temperatura, obtendríamos una curva continua, como podemos ver en la siguiente figura:

|  |
| --- |
| (\*\*\* gráfica página 4) |
| Gráfica de una magnitud analógica (temperatura en función del tiempo) (Fuente: libro Fundamentos Digitales) |

Ahora supongamos que medimos la temperatura cada hora (es decir, hacemos un muestreo). Lo que tenemos ahora son muestras que representan la temperatura en cada instante de tiempo (cada hora) a lo largo de un periódo de 24 horas. Si dibujáramos una gráfica de la temperatura con esta nueva forma de medición, obtendríamos una gráfica con valores discretos como la siguiente:

|  |
| --- |
| (\*\*\* gráfica página 5) |
| Gráfica de una magnitud analógica (temperatura en función del tiempo) (Fuente: libro Fundamentos Digitales) |

Hemos convertido una magnitud analógica a un formato (¡ojo, todavía no digital!) que puede digitalizarse, representando cada valor muestreado como un código digital a base de 0s y 1s. [23]

Cuando una señal analógica es muestreada y convertida a digital, utilizamos un elemento electrónico llamado conversor de analógico a digital (ADC, analogue-to-digital-converter). Cuando una señal digital se convierte en analógica, usamos un conversor de digital a analógico (DAC, digital-to-analogue-converter).

* 1. **Cicuitos Digitales**

(3:46) 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 tenemos 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.

|  |
| --- |
| http://obijuan.github.io/images/chip-mag-glass.png |
| El interior de un chip (Fuente: FPGA Wars) |

(5:05) Y da igual lo complejo que sea el chip, podemos coger el último Intel, el último ARM, no importa. 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 los **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 tenemos una memoria, con puertas lógicas podemos hacer multiplexores, unidades aritmético-lógicas (lo que vamos a hacer nosotros), etc. Todo va creciendo basándose en estos tres elementos. Entonces, combinándolos inteligentemente, podemos crear nuestros circuitos digitales. [20[

|  |
| --- |
| http://obijuan.github.io/images/digital-elements.png |
|  |
| Elementos básicos de los circuitos digitales (Fuente: FPGA Wars) |

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 expresares 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. (\*\*\* sacar esto de CODE). [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 (CIs) lineales (analógicos), los primeros CIs digitales basados en TTL (transistor-transistor logic) en los años 60, seguidos pronto por los CIs CMOS. Los primeros dispositivos incorporaban un número pequeño de puertas lógicas (nivel de expansión 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* (ROM, read-only) o de *acceso aleatorio* (RAM, random access). [20]
  1. **Tipos de circuitos digitales**

Los circuitos integrados digitales se dividen en dos categorías: lógica para funciones fijas y lógica programable.

* + 1. **Lógica para funciones fijas**

En lógica fija, las funciones lógicas son definidas por el fabricante y no se pueden modificar.

* 1. **Lógica programable**

(6:24) **El problema hasta ahora era la implementación.** Tú diseñabas un circuito digital en papel o en el ordenador, lo simulabas, pero luego a la hora de implementarlo **físicamente** con piecezitas reales pues era un latazo. **Y para eso vienen la lógica programables, que nos lo van a hacer muy fácil.**

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 (\*\*\* mostrar imagen de la matriz de componentes), es como “Conecta tú lo que quieras ahí”.

La idea con las es que inicialmente están **en blanco**, entonces decimos que no está **“configurado”**, están los elementos ahí sin conectar sin alimentar ni nada. Y de repente dices “Quiero hacer estas uniones” y te aparece el circuito. Es decir, **tenemos circuitos digitales bajo demanda.** Es como entrar en modo dios. “Quiero un controlador de VGA”, te aparece. “Quiero un microprocesador”, te aparece. “Quiero una ALU”, te aparece. Te los bajas o los diseñas tú, pero no has tenido que ir a la tienda a comprarlo, ¡lo tienes en la FPGA implementado físicamente!

|  |  |
| --- | --- |
| http://obijuan.github.io/images/fpga-config3.png | http://obijuan.github.io/images/fpga-config4.png |
| Primera configuración del PLD (Fuente: FPGA Wars) | Nuestro PLD reconfigurada con un circuito diferente (Fuente: FPGA Wars) (\*\*\* si te metes en la página, la imagen se refiere a FPGA, hmm… xD) |

Igual que en software podemos cambiar unas líneas de código por otras y ya tenemos un programa completamente diferente, podemos cambiar las conexiones que se hacen dentro de un circuito para tener un componente totalemente diferente. Para ello, como veremos, haremos uso de un software conectado a nuestro “hardware programable”, conocido como **Dispositivo Lógico Programable (PLD)**.

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 recablear 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.

Eso sí, si el circuito lo fabricásemos como un chip específico (ASIC, Application Specific Integrated Circuit) para nuestra aplicación, va a ser más óptimo, más rápido, etc. La FPGA no te va a dar tanto como si lo fabricases, 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 iniciales 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 quisiéramos 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 a tener muy en cuenta.

* 1. **Lógica programable versus procesadores (\*\*\* ¿En qué se diferencia de un microprocesador normal?)**

Cuando comencé a leer información por primera vez sobre PLDs y, en concreto, FPGAs, una de las cosas que más me costa entender era la diferencia entre programar un procesador (como haríamos con Arduino (\*\*\* comprobar que esto está bien) y “programar” un dispositivo lógico programable. Programar está escrito entre comillas porque, realemente, lo correcto sería “configurar”.

Un procesador es un dispositivo complejo con un conjunto **fijo** de instrucciones (lógica fija). Cada una de ellas está asociada a un hardware conectado de forma **fija** y el programador no puede usar más instrucciones que las definidas y configuradas por el fabricante. Sin embargo, por ejemplo, una FPGA no tiene nada conectado de forma fija, sino que está compuesta por una red o matriz de conexiones que el usuario puede **reconfigurar** uniendo o rompiendo conexiones, determinando el comportamiento lógico del dispositivo.

Además, cuando programamos un microprocesador, agrupamos una serie de instrucciones para su ejecución **secuencial** o en serie. Sin embargo, cuando configuramos una FPGA, el circuito resultante contendrá múltiples señales que variarán al mismo tiempo, en una especie de ejecución **paralela**. [3]

En resumen, la diferencia principal consiste en que cuando compramos un microprocesador, este ya tiene en su interior todos los circuitos lógicos “cableados” y nosotros no podemos cambiarlos, y por tanto, tampoco su funcionamiento. Sin embargo, si compramos nuestra FPGA, recibiremos un chip “en blanco” que podremos configurar a nuestro antojo usando software específico para ello y, además, las veces que queramos (para eso es un dispositivo lógico “programable”).

* 1. **ALU (Arithmetic Logic Unit)**

El procesador o CPU está formada 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 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 le dice a la ALU qué operación debe realizar y cuándo sobre un dato (\*\*\* chungo). 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. [14]

* + 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.

|  |
| --- |
| https://upload.wikimedia.org/wikipedia/commons/5/5e/JohnvonNeumann-LosAlamos.gif |
| John von Neumann (1903-1957) |

El ENIAC llamó la atención del matemático John von Neumann (1903-1957). Con reputación por hacer complejos cálculos aritméticos en su cabeza, ejercía de profesor en el Princeton Institute for Advanced Study, e investigó en todo desde mecánica cuántica hasta la aplicación de la teoría de juego 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 Designo f an Electronic Computing”, coautorizado con Arthur W.Burks t Herman H.Goldstine, describió varias características de un ordenador que supuso un avance respecto al ENIAC. Los diseñadores del EDVAC sentían 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 de programa y los datos mientras el programa se ejecutaba (el ENIAC no funcionoaba 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”. Aunque había una desventaja llamada el “cuello de botella de von Neumman” y es que una máquina con su 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. [21] Una mitigación frente a esto podría ser utilizar una memoria caché entre la CPU y la memoria principal. [31]

|  |
| --- |
| Resultado de imagen de von neumann architecture |
| Arquitectura de von Neumann |

* + 1. **Funcionamiento**

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 “crucen” 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**. [27]

|  |
| --- |
| https://upload.wikimedia.org/wikipedia/commons/c/c0/74181aluschematic.png |
| Circuitería de lógica combinacional del CI 74181, una ALU de 4 bits (Fuente: Wikipedia) |

En la figura anterior, vemos la circuitería de la primera ALU implementada en un único chip, esto es, obteniendo 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 anteriormente mostrada [32].

* + 1. **Operaciones**
       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. [27]

Para simplificar, en este trabajo nosotros vamos a implementar suma y resta, decremento e incremento.

* + - 1. **Operaciones lógicas**

A nivel lógico nos encontramos: AND, OR, XOR, complemento a uno. [27]

En este trabajo, implementaremos estas cuatro funciones.

* + - 1. **Operaciones sobre bits**

Normalmente son distintos tipos de desplazamientos: aritmético, lógico, rotación, todas tanto a izquierda como a derecha. [27]

Para simplificar, en este trabajo nosotros vamos a implementar el desplazamiento lógico a izquierda y derecha y la rotación a izquierda y derecha.

* 1. **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: A, A negada, B y B negada).

|  |  |
| --- | --- |
| \*\*\* Figura página 156 libro FD | \*\*\* Figura página 156 libro FD |
| Matriz AND no programada | Matriz and programada |

* 1. **Tecnologías de proceso basadas en conexiones programables**

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

* **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.

|  |
| --- |
| \*\*\* Figuras página 157 libro FD |
| Conexión programable mediante fusible |

* **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 el aislante pase a ser una conexión de baja resistencia, aplicando una tensión adecuada al antifusible o mantener este último intacto y mantener la conexión.

|  |
| --- |
| \*\*\* Figuras página 157 libro FD |
| Conexión programable mediante antifusible |

* **Tecnología basada en EPROM (*Electrically Programmable Read-Only Memory*).** 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 (OTP), a no ser que dispongan de un **encapsulado de ventana**. En este caso, la configuración se puede borrarse utilizando **luz ultravioleta (UV)** y volverse a programar de forma normal.

|  |
| --- |
| \*\*\* Figura página 158 libro FD |
| Tecnología basada en EPROM |

* **Tecnología basada en EEPROM (*Electrecally 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.**
* **Tecnología basada en SRAM.** 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 figura (b). 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 figura (c).

|  |  |
| --- | --- |
| \*\*\* Figura página 159 libro FD | \*\*\* Figura página 159 libro FD |
| Transistor on | Transistor off |
| Tecnología basada en SRAM | |

* 1. **Memoria de configuración**

Las tecnologías de proceso basadas en fusible, antifusible, EPROM y EEPROM son no volátiles, es decir, su configuración no se pierde cuando desconectamos el dispositivo de la alimentación. Sin embargo, la tecnología basada en SRAM sí es volátil, al quitar la alimentación perdemos culaquier configuración. Al alimentarlas de nuevo, están en blanco y se quedan esperando a recibir un bitstream para reconfigurarse.

Si nos centramos en las FPGAs, de las cuales hablaremos más adelante, contienen una **memoria serie externa**, no volátil, llamada **memoria de configuración**, que almacena el bitstream. Así, al alimentar la FPGA, lo primero que hace es reconfigurarse con el bitstream de la memoria de configuración.

El bitstream **se graba en la memoria de configuración** desde un dispositivo externo: ordenador, Arduino, Raspeberry Pi, etc.

(\*\*\* hablar más de la memoria de configuración. Hay más info en FPGA Wars, sección Introducción).

|  |
| --- |
| http://obijuan.github.io/images/fpga-mem-config-1.png |
| Memoria de configuración de la FPGA (Fuente: FPGA Wars) |

* 1. **Tipos de Lógica Programable**

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 (muchas puertas lógicas en su interior) que pueden reemplazar a miles de dispositivos de función fija. A continuación, veremos los principales:

* + 1. **SPLD (Simple Programmable Logic Device)**

El SPLD (Simple PLD, PLD simple) se corresponde con el PLD original y, generalmente, puede reemplazar a un número pequeño de CI de función fija. Los tres tipos principales de arquitectura SPLD son: PLA (Programable Logic array, Red de lógica programable), PAL (programable array of logic, red de lógica prgramable) y GAL (generic array of logic, red de lógica genérica):

* + - 1. **PAL (Programmable Logic Array)**

Un dispositivo lógico de matriz programable consta de una matriz programable de puertas AND que se conecta a una matriz fija de puertas OR. Con esto nos ahorramos el tiempo de propagación relacionado con el plano OR, produciendo un diseño más rápido. Sin embargo, perdemos flexibilidad a la hora de implementar del circuito.

|  |
| --- |
| \*\*\* Figura página 682 libro FD |
| Arquitectura PAL |

* + - 1. **GAL**

PLA y PAL son dispositivos programables una sola vez (OTP, one-time programmable) 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 (E2PROM) en lugar de fusibles y puede, por tanto, ser reconfigurada.

|  |
| --- |
| \*\*\* Figura página 684 libro FD |
| Arquitectura GAL |

* + - 1. **PLA**

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

|  |
| --- |
| \*\*\* Figura página 698 libro FD |
| Arquitectura PLA |

* + 1. **Complex Programmable Logic Device (CPLD)**

El CPLD es un paso más allá de los SPLDs; a partir de que los fabricantes fueron capaces de incluir más de un SPLD en un mismo chip, nació el CPLD o Complex PLD. Con él podemos crear diseños digitales más grandes.

Consiste en un número de bloques de matriz lógica (LAB, Logic Array Block) y una PIA (Programmable Interconnection Array). Cada LAB equivale aproximadamente a un SPLD. [23]

* + 1. **Field Programmable Gate Array (FPGA)**

La FPGA es un paso más allá en complejidad y densidad del CPLD pudiendo crear un diseño aún más grande. 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: el bloque lógico, las interconexiones programables y los bloques de entrada/salida (E/S). 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]

Los detalles de diseño específicos para cada FPGA dependen del fabricante. Por ejemplo, Xilinx utiliza los *bloques lógicos* como como un bloque lógico configurable (CLB) en sus FPGAs. El CLB está basado en una o más look-up tables (LUT) y biestables. El LUT está hecho de celdas de memoria (celdas SRAM). [20]

|  |  |  |
| --- | --- | --- |
| (\*\*\* diagrama página 27 libro Fundamentos Digitales) | (\*\*\* diagrama página 28 libro Fundamentos Digitales) | (\*\*\* diagrama página 29 libro Fundamentos Digitales) |
| Comparación entre distintos tipos de PLDs | | |

* 1. **Fabricantes de Lógica Programable**

Hay PLDs disponibles de una 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.

|  |
| --- |
|  |
| Fabricantes de lógica programable |

|  |
| --- |
|  |
| FPGA Altera Cyclone IV EP4CE115F23C8N |

|  |
| --- |
| Resultado de imagen de xilinx xc6slx25t |
| Xininx Spartan XC6SLX16 |

En este trabajo, vamos a trabajar con 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 Cliffor Wolf, que hizo ingeniería inversa y creó el proyecto Icestorm. El resto de FPGAs son privativas, 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 alquien 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 [12]:

|  |  |
| --- | --- |
| IceZUM Alhambra |  |
| Kéfir I |  |
| Nandland Go Board |  |
| Lattice Breakout Board |  |
| Lattice Icestick |  |
| Icoboard 1.0 |  |
| Placas con FPGAs libres | |

* 1. **Metodologias y herramientas de diseño de lógica programable**

Para diseñar con un PLD en particular, se requieren las herramientas de diseño apropiadas. Algunas son estas:

|  |
| --- |
|  |
| Herramientas de diseño de PLD por fabricante |

Estas herramientas nombradas en la tabla anterior son de carácter privativo, es decir, no conocemos los detalles internos de su funcionamiento (y, probablemente sean de pago). En este trabajo vamos a usar una de carácter libre, llamada 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. (13:42) Partimos de un **diseño**. Puede ser un dibujo en un papel o podemos tener una herramienta que nos haga el dibujo. A continuación, bien a mano o bien gráficamente, generamos un fichero de **descripción** hardware, siendo **Verilog** y **VHDL** los lenguajes de descripción más utilizados. Es como el HTML que describe la estructura de una página web, pues aquí describe la estructura del hardware. El siguiente paso es que a partir de tu descripción se hace la **síntesis** que es la parte complicada, 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, le enviamos este bitstream a la FPGA, es decir, se **configura**, y **mágicamente** aparece ahí nuestro circuito [0]. 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 cajitas).

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

**Síntesis del diseño:**

**Emplazado y enrutado (place and route):** mapear el diseño sintetizado a los recursos hardware del PLD. Con esto definimos qué partes del PLD contendrán qué funciones en el diseño y cómo las diferentes partes del PLD se interconectarán. 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 (\*\*\* el bitstream se genera en esta fase?):

|  |
| --- |
| http://obijuan.github.io/images/fpga-config8.png |
| Carga del bitstream en el dispositivo |

El bitstream se transmite por un bus serie (SPI), bit a bit, configurándose las conexiones del dispositivo [1].

**Post-layout delay extraction:** toma la información del diseño place and route y extrae retrasos de tiempo debido a las puertas lógicas y a las interconexiones usadas.

**Post-layout simulation:** usando los retrasos de tiempo, el diseño es simulado de nuevo con estos retrasos incluidos para determinar si el diseño todavía funciona correctamente.

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

**Configuración del PLD:** descarga los datos de configuración al PLD y habilita la configuración en el PLD para ser verificada en corrección.

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

Sea un CPLD o una FPGA lo que usemos, el diseñador sigue un flujo de diseño común para las etapas mayores en la entrada del diseño, verificación y configuración del dispositivo. Aún así, habrá sutiles diferencias entre el CPLD y la FPGA. La siguiente figura muestra un esquema simplificado del flujo de diseño típico de una FPGA.

|  |
| --- |
| https://cdn-images-1.medium.com/max/1000/0*fZcuYJ_MUJg3wgrE.png |
| Flujo de trabajo en una FPGA (Fuente: FPGA Wars) |

**Resumen: las FPGAs son #loMás.**

**Pila de herramientas libres**

(15:16) (\*\*\* coger imagen de FPGA Wars). La parte más importante es la de abajo. IceStorm es la que hizo Clifford, que es el que hace la ingeniería inversa a la placa. Iverilog para simular. GTKWave para visualizar esa simulación. Apio es una herramienta programada en Python que da soporte multiplataforma a estas tres herramientas anteriores. Son unos comandos que nos independizan del sistema operativo y puedes instalar y usar las herramientas de la capa anterior donde queramos. Sobre Apio tenemos dos herramientas: IceStudio permite hacer circuitos y pasarlos a la FPGA de una forma visual (es como el Scratch del hardware), ApioIDE es un editor de lenguaje de descripción de hardware basado en Atom con la que puedes diseñar por código, simular y sintetizar, teniendo todo a mano.

**IceStudio**

(17:00) (\*\*\* poner alguna captura de pantalla y un Hola Mundo, ahora hay un ejemplo demasiado complejo). El Hola Mundo del hardware es… “tirar un cable”.

|  |
| --- |
| https://raw.githubusercontent.com/FPGAwars/icestudio/develop/doc/images/main.png |
| Proyecto abierto en IceStudio (Fuente: FPGA Wars) |

**IceZUM Alhambra**

Nosotros trabajaremos con la placa IceZUM Alhambra. Tiene las siguientes caracterísiticas (\*\*\* traducir):

* FPGA development board ([iCE40HX1K-TQ144](https://github.com/Obijuan/open-fpga-verilog-tutorial/raw/master/tutorial/doc/iCE40LPHXFamilyDataSheet.pdf) from lattice)
* Open hardware
* Compatible with the [opensource icestorm toolchain](http://www.clifford.at/icestorm/), by Clifford Wolf
* Multiplatform: Linux / Mac / Windows
* Arduino like board: similar pinout than Arduino one / [BQ zum](https://store.bq.com/es/placa-zum-core).
* You can Reuse most of available Arduino / Zum shields
* Control your robots / printbots from an FPGA
* 12 MHZ MEMS oscillator
* ON/OFF switch (turn off your mobile robot easily)
* Input power voltage: 6 - 17v
* Max input current: 3A (Perfect for powering your robots)
* **20 Input/output 5v pins**
* 8 Input/Output 3.3V pins
* USB micro-B connector for programming the FPGA from the PC (same than zum board)
* FTDI 2232H USB device allows FPGA programming and UART interface to a PC
* Reset pushbutton
* 8 general purpose leds (user leds)
* 2 general purpose pushbuttons
* TX/RX leds
* Configuration in process led
* 4 analogue inputs though I2C bus
* Hardware protection against short-circuits, reverse polarity and so on

|  |
| --- |
| icezum-alhambra |
| Placa IceZUM Alhambra (Fuente: AlhambraBits) |

(\*\*\* poner una foto de mi placa).

Existen muchas placas como las de Xilinx, Altera o Papilio (algunas son libres y otras no) [12]. La Alhambra es una placa que utiliza una FPGA libre, la iCE40 de Lattice (Clifford Wolf fue la persona encargada del trabajo de ingeniería inversa a lo largo de tres años).

Ha sido diseñada por Eladio Delgado (\*\*\* link) en colaboración con Juan González (\*\*\* links), con la idea de que pudiera, además, ser utilizada en educación. También ha colaborado Jesús Arroyo, aportando IceStudio.

(\*\*\* no sé si esto es correcto) La empresa española BQ patrocinó el proyecto conjunto de la IceZUM Alhambra junto con IceStudio y Apio en sus inicios a lo largo de 2016 y ahora el proyecto avanza gracias a sus creadores con el apoyo de la comunidad gracias a su concepto abierto y colaborativo.

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 (\*\*\* links). Incluso el propio Juan González, también conocido como Obijuan, ha creado una “Academia Jedi de Hardware”: cada cierto tiempo sube un tutorial sobre la IceZUM Alhambra, al final de ese tutorial propone unos ejercicios, los usuarios los resuelven y los suben a redes y al repositorio de Github para conseguir “bitpoints”. Una gran iniciativa para animar a explorar en las FPGAs y para fomentar la competitividad. (\*\*\* links)

|  |
| --- |
| https://github.com/Obijuan/Entregas-Tutorial-Electronica-Digital-FPGAs/raw/master/Tutorial-3/lobotic/lobotic-diploma.jpg |
| Academia Jedi de Hardware (Fuente: FPGA Wars) |

**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, el segundo mayor fabricante de FPGAs mundial, por parte de Intel en 2015. La entrada en escena de este último a incrementado el nivel de competencia, provocando el lanzamiento de productos basados en FPGA con mayores capacidades y a menor precio.

En 2016, Intel presentó sus procesadores para servidores Xeon con circuitería FPGA integrada, destinados al diseño de centros de procesamientos de datos (CPD) que, como los creados por Microsoft para sus servicios Azure y Bing a partir de su proyecto Catapult, combinan un microprocesador clásico con la posibilidad de implementar en hardware las partes más críticas para el rendimiento y consumo del sistema.

Por otra parte, Xilinx, el primer fabricante mundial de dispositivos FPGA, también cuenta con productos que aúnan en un mismo encapsulado hardware reconfigurable con núcleos de procesamiento tipo ARM. Ambas gamas de producto, la de Intel y la de Xilinx, hacen posible la implementación de soluciones híbridas hardware-software. Esta es una tendencia que está llegando incluso a los servicios de computación en la nube, como los 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. Todo ello apunta a un gran futuro de esta tecnología. [8]

No ha sido hasta ahora que las FPGAs se están empezando a popularizar. **Se trata de una tecnología muy cerrada**, rodeada de software privativo, en la que solo puedes usar lo que el fabricante te dice en las condiciones que te dice. 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 la primera toolchain (\*\*\* aclarar qué es esto) de herramientas que permiten pasar de Verilog al bitstream usando solo herramientas libres. [1]

**Hardware reconfigurable y FPGAs (\*\*\* un timeline estaría chachi (pero más arriba))**

Vamos a echar un vistazo a la historia de los dispositivos de lógica programable (PLDs) para tener una perspectiva sobre ellos, PLDs en general y field programable gate arrays o FPGAs en particular. Las FPGAs son un subconjunto de los PLDs. Aquí un timeline.

|  |
| --- |
|  |
| Timeline del hardware reprogramable (Fuente: Coursera) |

El deseo de tener hardware programable ha existido desde los comienzos del hardware digital.

En la década de los 80 el fabricante Lattice introduce los GAL (Generic Array Logic), análogos a los PAL pero con una tecnología de establecimiento de las conexiones basada en memoria EEPROM. Al contrario que los dos primeros, los GAL podrían ser reconfigurados tantas veces como se necesitase, aportando mucha más flexibilidad.

Actualmente, los PAL, PLA y GAL están en desuso. En su lugar se recurre a los CPLD (Complex Programmable Logic Device) y FPGA. El encapsulado de estos deja atrás el clásico DIP (Dual In-Line Package) con unas pocas decenas de pines, mientras que la tecnología de configuración pasa a estar basada en memoria tipo Flash o bien SRAM (Static RAM). La mayor ventaja respecto a los CI ya mencionados estriba en la mayor densidad de elementos básicos reconfigurables, pasando de pocas decenas de puertas lógicas a miles o millones de ellas. Además, no es necesario disponer de hardware especializado para establecer la configuración de estos CI, esta se obtiene directamente de la memoria del ordenador o de la placa de entrenamiento correspondiente.

1. **Detalles del proceso**

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

En concreto, 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.

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, que 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. (\*\*\* la parte de expresar los circuitos de forma matemática no la he hecho, tipo suma de productos y producto de sumas).

Al pasarlos a Logisim (existen más simuladores, pero he escogido este por su presencia en Internet y foros, en definitiva, por soporte) 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.

* + **Sumador de 4 bits (4-Bit Full Adder)**

(\*\*\* ver Code) Para realizar el sumador, vamos a descomponerlo en unidades más pequeñas, half-adders de 1 bit. Con dos half-adders de 1 bit tenemos un full-adder de 1 bit, que nos permite tener una entrada de acarreo (sobre todo útil cuando vamos a sumar números de más de 1 bit):

|  |
| --- |
|  |
| 1-Bit Half Adder |

|  |
| --- |
|  |
| 1-Bit Full Adder |

Con 4 Full Adders de 1 bit temenos un Full Adder de 4 bits, con el que podremos sumar números de 4 bits:

|  |
| --- |
|  |
| 4-Bit Full Adder |

* + **Restador de 4 bits (4-Bit Subtractor)**

Para hacer el restador, usamos el Full Adder anterior con dos diferencias: la entrada Carry In está a 1 (veremos por qué) y la entrada B es invertida. Esto es porque utilizamos 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. (\*\*\* buscar más sobre el complemento a dos)

* + Incrementador de 4 bits (4-Bit Incrementer)

El incrementador está hecho con Half Adders de 1 bit, y lo único especial es que recibe un Carry In con valor a 1, ya que esa va a ser siempre su función: sumar uno al dato de entrada.

|  |
| --- |
|  |
| Incrementador de 4 bits (4-Bit Incrementer) |

* + **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 (es algo parecido a hacer un módulo).

|  |
| --- |
|  |
| Decrementador de 4 bits (4-Bit Decrementer) |

* + **Desplazamiento a la izquierda de 4 bits (4-Bit Left Shift)**

El MSB se pierde. (\*\*\* detallas mejor)

|  |
| --- |
|  |
| Desplazamiento a la izquierda de 4 bits (4-Bit Left Shift) |

* + **Desplazameinto a la derecha de 4 bits (4-Bit Right Shift)**

Lo único que cambia respecto al desplazamiento a la izquierda es que la entrada Left Shift está a 0 y la entrada Rigth Shift a 1.

El LSB se pierde.

|  |
| --- |
|  |
| Desplazameinto a la derecha de 4 bits (4-Bit Right Shift) |

* + **Rotación a la izquierda de 4 bits (4-Bit Left Rotate)**

Es un desplazamiento a la izquierda con la diferencia de que el MSB se pone en la posición del LSB.

|  |
| --- |
|  |
| Rotación a la izquierda de 4 bits (4-Bit Left Rotate) |

* + **Rotación a la derecha de 4 bits (4-Bit Right Rotate)**

Es un desplazamiento a la derecha con la diferencia de que el LSB se pone en la posición del MSB.

|  |
| --- |
|  |
| Rotación a la derecha de 4 bits (4-Bit Right Rotate) |

* + **AND de 4 bits (4-Bit AND)**

Es más fácil porque implementar la función lógica AND tiene mapeo directo a la puerta lógica.

|  |
| --- |
|  |
| AND de 4 bits (4-Bit AND) |

* + **OR de 4 bits (4-Bit AND)**

Igual que el AND de 4 bits.

|  |
| --- |
|  |
| OR de 4 bits (4-Bit AND) |

* + **XOR de 4 bits (4-Bit OR)**

Igual que el AND de 4 bits.

|  |
| --- |
|  |
| XOR de 4 bits (4-Bit OR) |

* + **Complemento a uno de 4 bits (4-Bit One’s Complement)**

|  |
| --- |
|  |
| Complemento a uno de 4 bits (4-Bit One’s Complement) |

Igual que el AND de 4 bits. En este caso, se utilizan puertas lógicas NOT.

* + **Multiplexor de 16 a 4 (16-To-4 Multiplexer)**

|  |
| --- |
|  |
| Multiplexor de 16 a 4 (16-To-4 Multiplexer) |

* + **Multiplexor de 4 a 4 (4-To-4 Multiplexer)**

|  |
| --- |
|  |
| Multiplexor de 4 a 4 (4-To-4 Multiplexer) |

* + **ALU de 4 bits (4-Bit ALU)**

Unimos todos los componentes

|  |
| --- |
|  |
| ALU de 4 bits (4-Bit ALU) |

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

[15]

Abrimos IceStudio.

Ponemos todos los componentes y los conectamos entre sí.

Vamos a *Edit > Project Information* y rellemanos los datos según nuestras preferencias.

|  |
| --- |
|  |
| Edit > Project Information |

|  |
| --- |
|  |
| Edit > Project Information |

Guardamos el projecto como un archivo *.ice* normal.

Para utilizarlo en otro componente, simplemente nos vamos a *File > Add as Block* y lo arrastramos al canvas.

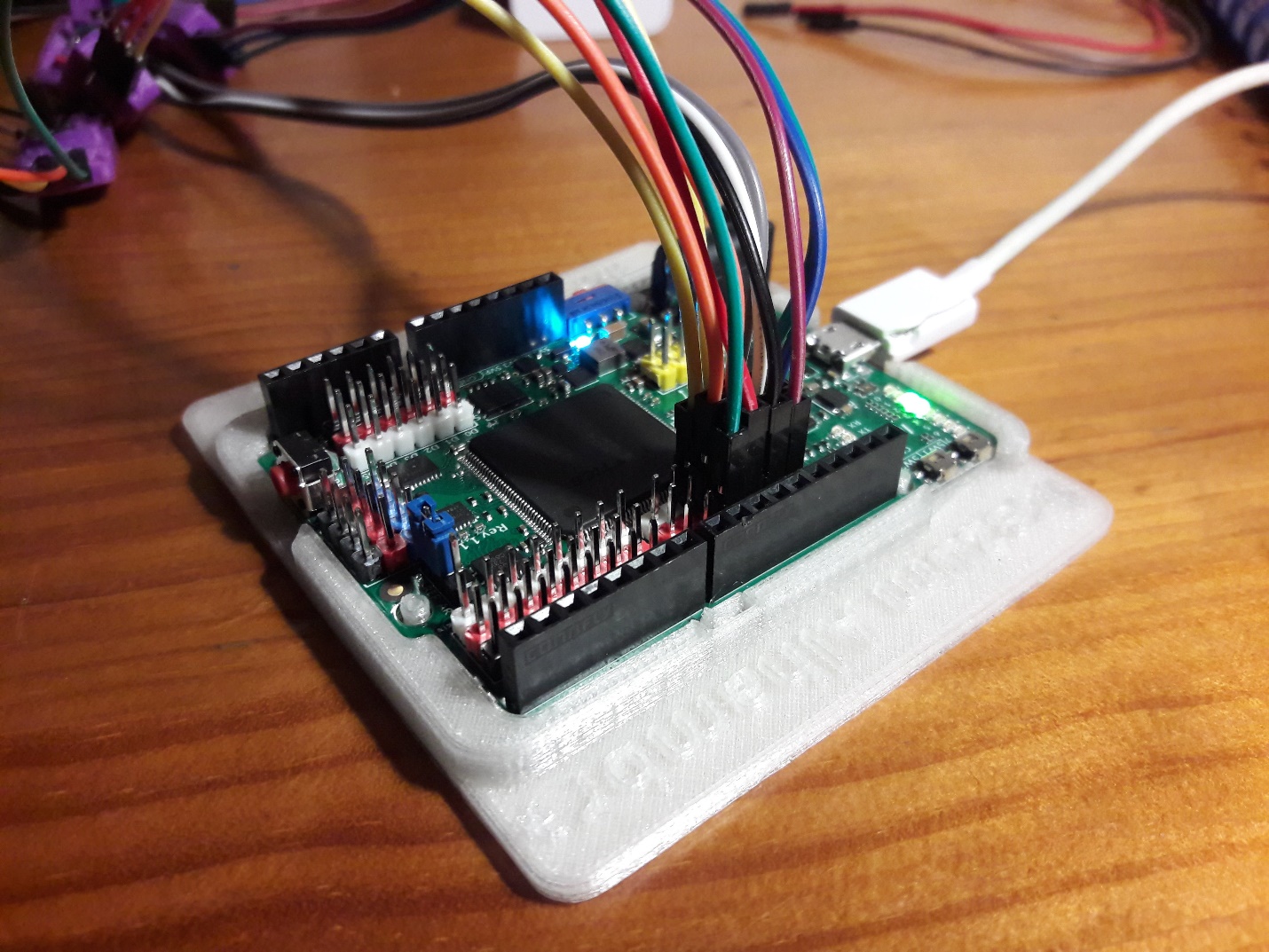
**3.3. Etapa 3: pruebas en la FPGA**

Tenemos que configurar las salidas de los módulos que vayamos a probar por salidas que tengan mapeo en la FPGA. Para ello hacemos doble click sobre la salida y marcando “FPGA pin”:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Antes de configurar la salida | Configurando de la salida como pin de la FPGA | Después de configurar la salida |

Solo nos queda enchufar los interruptores a la FPGA y cargar el circuito en ella:

|  |
| --- |
|  |
| Interruptores enchufados y circuito (4-Bit Rigth Rotate) cargado en la FPGA |



**DIY**

**Compra de materiales (\*\*\* listar)**

* Interruptores
* LEDs
* Cables (hembra-hembra)
* Resistencias

**Impresión piezas 3D (\*\*\* poner planos)**

La comunidad FPGA Wars tiene un repositorio con piezas 3D que se pueden usar para usar componentes electrónicos de forma más compacta. En este caso, vamos a usar Alhambra-Switch [19], una forma de juntar interruptor cables y conector en una sola pieza 3D y poder trabajar con él con mayor comodidad.

Si necesitas modificar la pieza porque, por ejemplo, tu interruptor sea diferente, siempre puedes editar la pieza en algún programa CAD. Hay opciones con licencia libre o como en este caso, usar SketchUp de Google, del cual te puedes descargar una versión de prueba. Este último es muy fácil de usar.

|  |
| --- |
|  |
| Modelo en 3D 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:

|  |  |
| --- | --- |
|  |  |
| Extension Warehouse | El plugin se llama SketchUp STL |

Una vez tenemos el plugin instalado, para exportar, nos vamos a *File > Export STL* y cuando aparezca el cuadro de diálogo, configuramos según nuestras preferencias:

|  |  |
| --- | --- |
|  |  |
| File > Export STL | En el cuadro de diálogo configuramos según nuestras preferencias |

Solo queda imprimir las piezas 😊.

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

Hay varias formas de hacerlo. Yo estuve probando varias y al final me quedé con la que me parecía más cómoda.

*Nota: depende de las características del interruptor habrá que soldar de una forma o de otra. Por ejemplo, hay interruptores deslizantes que tienen seis patillas en vez de tres. También hay que tener cuidado de conectar los pines del interruptor con los pines del conector correctos, ya que si no, puede 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). Mira bien la hoja de especificaciones.*

|  |  |
| --- | --- |
|  |  |
| 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) | Lo metemos en la pieza 3D en la posición adecuada (si no, puede que luego funcione al revés) |
|  |  |
| Colocamos los cables en sus posiciones para prepararlos para soldarlos | Soldamos |
|  |  |
| Así queda una vez soldado | Y aquí tenemos nuestros ocho interruptores terminados |

**Realización de PCB**

1. **Conclusiones**

Gracias al descubrimiento de divulgadores como Juan González he confirmado que lo técnico no tiene por qué ser algo difícil de aprender, al contrario, si se explica bien, se entiende bien. Lo único, que hay que contar con los medios necesarios y metodologías que te motiven a seguir.

Aunque el objetivo final no sea este, gracias a la investigación previa para poder obtener el resultado final (la implementación de la ALU en la FPGA), he conseguido ampliar mi visión global sobre la informática, en el sentido de aprender cómo se complementan hardware y software entre sí.

1. **Glosario**

FPGA

Microprocesador

Chip

Transistor

Silicio

Intel

ARM

Puerta lógica

Biestable

Cable

Registro

**6. Referencias**

[0] Primer vídeo que vi de Obijuan: <https://www.youtube.com/watch?v=XWC1B7UKv98>

[1] <http://fpgawars.github.io/>

[2] <https://github.com/FPGAwars>

[3] <https://planetachatbot.com/qu%C3%A9-es-una-fpga-y-por-qu%C3%A9-jugar%C3%A1n-un-papel-clave-en-el-futuro-e76667dbce3e>

[4] <http://www.fpgadeveloper.com/2011/07/list-and-comparison-of-fpga-companies.html>

[5] <https://juegosrobotica.es/icezum-alhambra-educacion/>

[6] <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial>

[7] Esto está muy bien, muy detallado, sobre FPGAs, Verilog, VHDL, etc:

<http://repository.upb.edu.co/bitstream/handle/20.500.11912/837/digital_19163.pdf?sequence=1&isAllowed=y>

[8] Esto también está muy bien, muy detallado, mucho sobre la historia y el estado actual de las FPGAs:

<http://sinbad2.ujaen.es/sites/default/files/publications/FPGA.pdf>

[9] Curioso: <http://www.digibarn.com/collections/parts/mac-wirewrap5-board/>

[10] Del IEEE: <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7086413>

[11] Diferencia entre los diferentes dispositivos lógicos programables (PLDs): <https://es.coursera.org/learn/intro-fpga-design-embedded-systems/lecture/YaCfa/2-a-brief-history-of-programmable-logic>

[] Esquema dispositivos lógicos programables (otro diferente al que ya tenemos): <http://www.eng.auburn.edu/~nelson/courses/elec4200/Slides/Programmable%20Logic%20Devices.pdf>

[] Otro sobre dispositivos lógicos programables y más cosas genéricas de lógica: <http://www.eng.uwi.tt/depts/elec/staff/lngalamou/EE19D/ee19d-lect5.pdf>

[] Puede ayudar en lo de los PLDs: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.7513&rep=rep1&type=pdf>

[12] Distintas placas: <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki/V%C3%ADdeo-3:-La-Icezum-Alhambra-y-otras-placas-con-FPGAs-libres>

[13] Qué sucede en la FPGA (visualmente): <https://github.com/Obijuan/digital-electronics-with-open-FPGAs-tutorial/wiki/V%C3%ADdeo-4:-Mi-primer-circuito.-Encendiendo-un-led>

[14] Qué es y para qué sirve una ALU (Quora): <https://www.quora.com/What-does-ALU-do-in-computer>

[15] <https://sites.google.com/site/angmuz/proyecto-f3---haciendo-bloques-con-icestudio>

[16] PCB basics (ver enlaces del final, son muy interesantes): <https://learn.sparkfun.com/tutorials/pcb-basics>

[17] Cómo fabricar cloruro férrico casero: <https://www.youtube.com/watch?v=2Vggcs5Kicg>

[18] Cómo hacer un PCB casero: <http://www.instructables.com/id/Making-A-Customized-Circuit-Board-Made-Easy/>

[19] Alhambra-Switch: <https://github.com/PCBPrints/Alhambra-switch/wiki>

[20] Libro PLD rojo escaneado

[21] Libro CODE: <https://www.amazon.com/Code-Language-Computer-Hardware-Software/dp/0735611319>

[22] Extras del libro CODE (señales de control): <http://www.charlespetzold.com/code/CodeTechnicalAddendum.html>

[23] Libro Fundamentos Digitales:

[24] FPGA Altera Cyclone IV EP4CE115F23C8N: <https://www2.hdl.co.jp/en/altera-series1/cyclone-ql/acm-006-ql.html>

<https://www.buyaltera.com/PartDetail?partId=2260176>

[25] FPGA Xilinx Spartan : <https://www.ztex.de/imgs/fpga-2.00-1600.jpg>

<https://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html#productTable>

[26] Lattice ICE40HX1K-TQ144: <http://www.latticestore.com/products/tabid/417/categoryid/9/productid/560/searchid/1/searchvalue/ice40hx1k-tq144/default.aspx>

[27] ALU Wikipedia: <https://en.wikipedia.org/wiki/Arithmetic_logic_unit>

[28] Página interesante con circuitos digitales y explicaciones: <http://www.play-hookey.com/digital/>

[29] Motorola ALU datasheet: <http://www.esi.uclm.es/www/isanchez/apuntes/ci/74181.pdf>

[30] ALU 74181: <https://en.wikipedia.org/wiki/74181>

[31] Arquitectura de von Neumann: <https://en.wikipedia.org/wiki/Von_Neumann_architecture>

[32] ALU 74181 Signetics datasheet: <http://pdf.datasheetcatalog.com/datasheets/560/493318_DS.pdf>

[33] Lattice ICE40HX1K-TQ144: <https://www.mouser.es/datasheet/2/225/iCE40LPHXFamilyDataSheet-1022803.pdf>