**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>

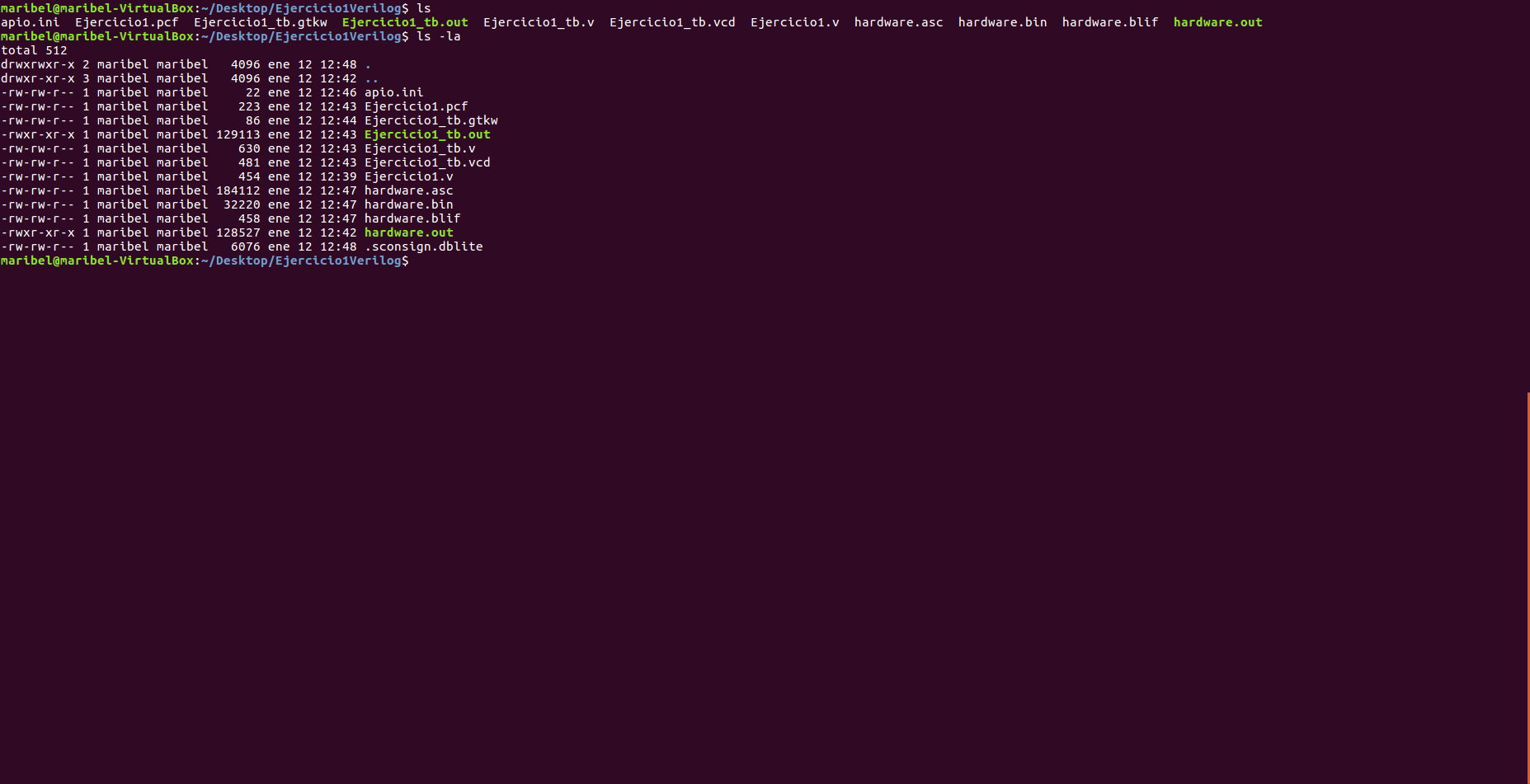
Esto es TOP: <https://www.allaboutcircuits.com/textbook/digital/chpt-16/microprocessor-programming/>

*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.

**Objetivo**

Implementar una ALU en una FPGA.

**Introducción**

(3:46) Si vemos el interior de los chips digitales 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 unas cosas 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 estas cosas tan sencillas.** Si te pones a unir biestables tienes un registro, si unes varios registros tienes una memoria, con puertas lógicas puedes hacer multiplexores, unidades aritmético-lógicas (lo que vamos a hacer nosotros), etc. Todo va creciendo basándose en estos tres elementos (increíble, ¿no?). Entonces, combinándolos inteligentemente, podemos crear nuestros circuitos digitales. (\*\*\* poner imágnes de FPGA Wars).

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

Pero puedes pensar, sí, la electrónica digital son tres cosas básicas, pero es muy difícil, es un rollo, no me gusta. Pues investigando en Internet, encontré un juego del móvil llamado “Circuit Scramble” y es Electrónica Digital, son circuitos combinacionales, algunos secuenciales 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. (\*\*\* poner captura de la pantalla).

**¿Qué es una FPGA?**

(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 las FPGAs, que nos lo van a hacer muy fácil.**

La FPGA 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”.

(6:59) Y la idea con las FPGAs es que inicialmente están **en blanco**, entonces decimos que no está **“configurada”**, están los elementos ahí sin conectar sin alimentar ni nada. Y de repente dices “Quiero hacer estas uniones” y te aparece el circuito. Las uniones de las FPGAs son **reconfigurables**, tú con bits, puedes decidir que una unión se haga y otra no, a tu gusto. Lo interesante de esto es que dices “Ahora ya no quiero este circuito, quiero otro”. Bueno, pues estableces otra configuración y te aparece tu nuevo 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 de la FPGA (Fuente: FPGA Wars) | Nuestra FPGA reconfigurada con un circuito diferente (Fuente: FPGA Wars) |

(\*\*\* hablar más de la reconfiguración de la FPGA. Hay más info en FPGA Wars, sección introducción).

(8:10) Por hacer una analogía, las FPGAs son como las impresoras 3D de los circuitos digitales. ¿Quieres algo? Pues lo diseñas y lo imprimes. En la FPGA. ¿Quieres algo? Lo diseñas y lo configuras la FPGA para que lo implemente.

Cons: Lo que podemos implementar en la FPGA está limitado por el espacio, realmente, por las 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 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**.

(8:43) E igual que con las impresoras 3D, ahora los diseños de los circuitos se convierten en software. Veamos esto. Ahora manejamos las mismas herramientas que en software (editores, repositorios, etc.) de manera que puedo **compartir** el hardware. Entonces, podemos diseñar un circuito, lo subimos al repositorio, cualquier persona se lo baja, lo **sintetiza** en la FPGA y le aparece. (\*\*\* esto es más sobre lo libre, hablar más de esto)

**Resumen: las FPGAs son la hostia.**

**¿En qué se diferencia de un microprocesador normal?**

Un procesador es un dispositivo complejo con un conjunto **fijo** de instrucciones. 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, 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.

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**.

Podemos verlo de otra manera: acelerar el ritmo de trabajo de un microprocesador se traduce en más ciclos de reloj, es decir, más tiempo pero en el mismo espacio (secuencialidad). Sin embargo, acelerar el ritmo de trabajo de una FPGA se traduce en más circuitería, es decir, más espacio pero en el mismo tiempo (paralelismo).

**Nota:** si tenemos a mano el diseño (como el plano de una casa) de un microprocesador, y si nuestra FPGA tiene los recursos necesarios, podríamos implementar ese microprocesador en la FPGA. Esto es lo que se suele llamar **“soft microprocessor”** (no tenemos un chip fabricado exclusivamente como microprocesador, pero tenemos una red de elementos electrónicos y cables que, digamos, nos lo simula (\*\*\* ver parte de optimización y rendimiento en la introducción)). De hecho, en la vida real, se suele recurrir a FPGAs como un paso previo a la fabricación final, en una especie de proceso de prototipado.

**Flujo de trabajo**

(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 automáticamente (\*\*\* ver IceStudio), 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.

(\*\*\* hablar más del bitstream. Hay más info en FPGA Wars, sección Introducción)

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

**Memoria de configuración**

Por cierto, las FPGAs son **volátiles**: al quitar la alimentación pierden su configuración. Al alimentarlas de nuevo, están en blanco y se quedan esperando a recibir un bitstream para reconfigurarse.

Por eso, la FPGA contiene 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) |

**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 alguan 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) |

**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**

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 primer PLD fue un PROM o Programmable Read Only Memory fue inventado en 1956 pero no estuvo disponible comercialmente hasta 1969. Seguido rápidamente por EPROMs o Erasable Programmable Read Only Memory in 1971. Y entonces, PLAs o Programmable Logic Arrays en 1975. Y PALs o Programmable Array Logic en 1978. Veremos la diferencia (\*\*\* buscar sobre esto).

El deseo de tener hardware programable ha existido desde los comienzos del hardware digital. (\*\*\* ahora viene la parte de Sum Of Products y Products Of Sums, de la que no me acuerdo NADA).

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

Al final: FPGAs < PLDs < Logical Devices [11]

(\*\*\* pasar esquema a limpio)

Los circuitos integrados tradicionales, con el microprocesador al frente de una familia de la que también forman parte microcontroladores, ASIC, DSP (Digital Signal Processing) y otros, se caracterizan por contar con una microarquitectura hardware **fija** (\*\*\* esto es lo mismo que en la introducción). La función de este tipo de circuitos se establece durante su diseño y queda fijada de forma permanente durante el proceso de fabricación. Algunos circuitos integrados (CI), como es el caso del microprocesador, son de propósito general. Por ello incorporan la capacidad de ser programados (que no reconfigurados) una y otra vez, ejecutando funciones definidas por software. Otros, como los ASIC y DSP, se diseñan desde un principio para satisfacer una función específica, siendo menos flexibles que un microprocesador pero ofreciendo a cambio un mejor rendimiento, mayor eficiencia energética, entre otras ventajas.

Paralelamente a los anteriores, durante las últimas tres décadas también se han desarrollado CI configurables por el usuario. Inicialmente estas soluciones, plasmadas en circuitos como las PAL y PLA, se basaban en **tecnologías de fusibles**, por lo que únicamente podían configurarse una vez. La disponibilidad de nuevos tipos de memorias, incluyendo las estática y las no volátiles, han hecho posible fabricar CI reconfigurables.

Los PAL y PLA ofrecen matrices de puertas lógicas con conexiones sin establecer, de forma que el CI puede realizar diferentes funciones según la aplicación que se le quiera dar. Las conexiones no se cablean, sino que consiste en “quemar” unos fusibles internos usando para ello un hardware a medida, como podía ser una placa conectable a un PC.

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.

**ALU**

Representar y almacenar números es una función importante en un ordenador, pero el objetivo real es la computación o manipulación de números de una forma estructurada y útil, como puede ser la suma de números. La ALU es el cerebro matemático de un ordenador. Cuando comprendes el diseño y funcionamiento de la ALU, entiendes una parte fundamental de los ordenadores modernos. Es LA cosa que hace todos los cálculos en un ordenador, así que básicamente todos lo utilizan.

Es un circuito digital usado para realizar **operaciones aritméticas y lógicas**. 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]

Nosotros vamos a construir una ALU con 12 (\*\*\* volver a contar) bloques que realizan diferentes operaciones: suma, resta, desplazamiento a izquierda, desplazamiento a derecha, incrementador, decrementador, rotación a izquierda, rotación a derecha, AND, OR, XOR y complemento a uno. 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.

Las operaciones lógicas son sencillas de implementar, ya que se mapean directamente en puertas lógicas con su mismo nombre. El resto de operaciones son un poco más complejas. Las iremos viendo.

**Detalles del proceso**

Comencé investigando sobre las FPGAs. Di con un vídeo de Obijuan en la que explicaba de forma muy sencilla y amena qué eran las FPGAs, para qué servían y cómo se usaban (ver ¿Qué es una FPGA?). Así que decidí meterme en la comunidad y descubrí que tenían una placa con una FPGA llamada IceZUM Alhambra. Tras ver todo el movimiento y actividad que había, me compré una. Seguí los tutoriales de Obijuan para ir aprendiendo. Al principio 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, en este caso una ALU. Hacer una ALU sencilla puede que sea la típica práctica de primero de carrera, pero lo más interesante para mí era conocer qué era eso de las FPGAs y cómo implementar mi ALU en ella. Luego conocí un libro llamado “Code”, muy bueno, que te explica de manera sencilla y amena la historia de la computación y un viaje desde los relés, pasando por los transistores hasta construir puertas lógicas, biestables, memorias y procesadores y circuitos integrados, así como la invención del lenguaje ensamblador y de los lenguajes de alto nivel, hasta el funcionamiento de las pantallas. Decidido qué hacer y habiendo comenzado el diseño de la ALU, me puse a imprimir piezas 3D para formar interruptores que pudiera conectar a la FPGA para que me sirvieran como entrada de datos.

* **Etapa 1: diseño de la ALU en Logisim (simluador)**

Para ello, 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) |

**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.

**Etapa 3: pruebas en la FPGA**

Glosario

FPGA

Microprocesador

Chip

Transistor

Silicio

Intel

ARM

Puerta lógica

Biestable

Cable

Registro

Referencias:

[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: <https://es.coursera.org/learn/intro-fpga-design-embedded-systems/lecture/YaCfa/2-a-brief-history-of-programmable-logic>

[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/>