

Un saludo, no trato de reinventar la rueda ni decir que yo la invente, pero si estamos rediseñándola para que pueda ser utilizada por otras personas en diferentes plataformas. En este caso estoy realizando este tutorial para los que usamos Mac.

Este manual esta hecho hasta esta fecha para la version 10.11.4 de Mac (Capitan).



Gracias a la ayuda de Juan Gonzales y los pasos a seguir que el me ofrecia pude realizar los 3 pasos desde la creacion del codigo en Verilog -> la compilacion y simulacion -> hasta la implementacion en el fpga.

Las pruebas de la implementacion en el fpga fueron realizadas en la placa **Lattice – ice40**



En primer lugar al encontrar un proyecto libre de hardware para la implementacion de nuestros propios disenos me parecia algo extraordinario en el cual siendo estudiante de ingenieria me parecia algo a seguir.

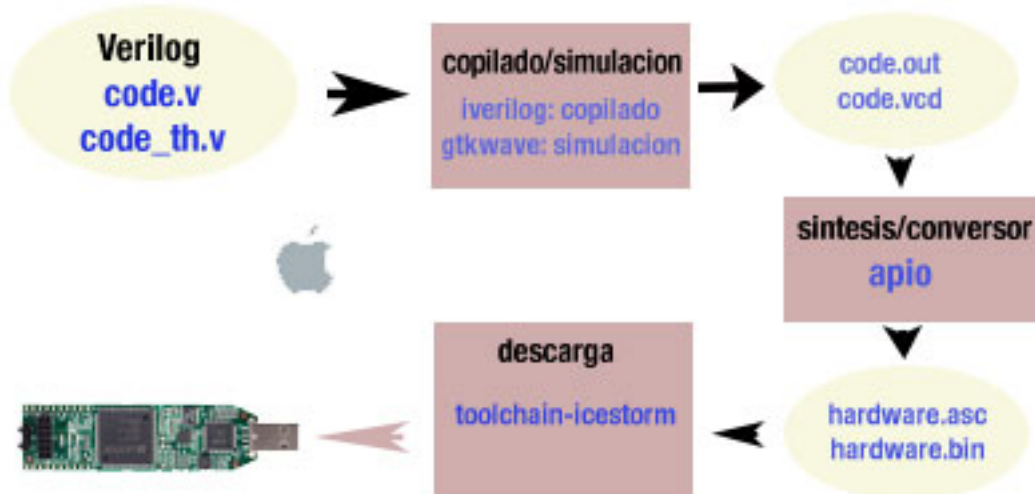
Pero al ver que la mayoría de aplicaciones estaban disenadas para windows o linux a veces se hacia un poco complicado, no la instalacion del SO sino mas bien el consumo de recursos para personas que tengas en este caso una MAC ya que se necesitaba crear una VM con windows y para los que odiamos MS creamos una Linux.

Cabe recalcar que tengo experiencia mas en VHDL en lo cual investigando encuentre la manera de escribir codigo VHDL y copilarlo y simularlo en mac utilizando de igual manera herramientas open source. Pero ahora me voy enfocar en este manual paso a paso con las herramientas de Verilog.

3-PASOS

Los circuitos se diseñan utilizando un **lenguaje de descripción hardware** (HDL), como Verilog o VHDL. Son los ficheros fuentes. La generación del bitstream se hace en dos fases, a partir de las fuentes:

En la siguiente figura se muestran las diferentes herramientas usadas en las etapas, y las extensiones de los archivos que se van generando: (nota: durante este proceso va ser diferente al proceso original ya que utilizaremos otras herramientas que nos ayudaran a la sintesis/conversion y a la descarga al FPGA).



PASO 1

Herramientas libres para simulación

Para diseñar los circuitos es fundamental disponer de un simulador de verilog. Las herramientas libres que usaremos son:

- **Simulador de Verilog:** [Icarus Verilog](#)
- **Visualizador de señales:** [Gtkwave](#)

Icarus verilog crea un fichero ejecutable a partir del código Verilog. Al ejecutarlo se realiza la simulación. Los resultados se vuelcan a un **fichero .vcd** que se visualiza con la herramienta **Gtkwave**. Esto nos permite inspeccionar las señales para comprobar su correcto funcionamiento.

La compilación y simulación de manera manual después de instalar las herramientas de simulación es como el ejemplo a continuación:

Verifica la version

```
$ iverilog -V  
Icarus Verilog version 0.9.7 (v0_9_7)
```

Copila el archivo fuente y el código para la simulación, creando un ejecutable .out

```
$ iverilog -o code.out code.v code.v
```

Verifica la creación del archivo .out

```
$ ls -la *.out  
-rwxr-xr-x 1 xxxxxx xxxxxxxx 1034 mar 20 09:16 setbit.out
```

Ahora ejecuta el setbit.out:

```
$ ./setbit.out  
VCD info: dumpfile setbit_tb.vcd opened for output.  
Componente ok!
```

Esto te crea el fichero setbit_tb.vcd con los resultados de la simulación

Y el archivo lo puedes abrir con el gtkwave:

```
$ gtkwave setbit_tb.vcd setbit_tb.gtkw
```

GTKWave Analyzer v3.3.66 (w)1999-2015 BSI.

PASO 2

Herramientas libres para la synthesis y conversion.

En este paso vamos a utilizar otra herramienta llamada Apio. Que es tambien un proyecto experimental basado en [platformio](https://github.com/bqlabs/apio).

Pueden encontrar el repositorio en: <https://github.com/bqlabs/apio>

NOTA: deben tener instalado en su mac [Python 2.7](https://www.python.org/) y [Homebrew](https://brew.sh/) antes de cualquier otra instalacion.

Una vez instalado el homebrew instalas el siguiente paquete.

brew install libftdi0

Una vez instalado todo procedemos a ejecutar dentro de nuestra misma carpeta donde se encuentra nuestro proyecto inicial el siguiente commando:

\$> apio build

```
== setbit ==  
  
Number of wires:          1  
Number of wire bits:      1  
Number of public wires:   1  
Number of public wire bits: 1  
Number of memories:       0  
Number of memory bits:    0  
Number of processes:      0  
Number of cells:          0  
  
2.26. Executing CHECK pass (checking for obvious problems).  
checking module setbit..  
found and reported 0 problems.  
  
2.27. Executing BLIF backend.  
  
End of script. Logfile hash: 8a42f63061, CPU: user 0.27s system 0.02s  
Yosys 0.6 (Apio_build_0.6)  
Time spent: 55% 8x read_verilog (0 sec), 17% 1x share (0 sec), ...  
arachne-pnr -d 1k -o hardware.asc -p setbit.pcf hardware.blif  
icepack hardware.asc hardware.bin
```

El cual nos creara dos archivos: uno .asc y otro .bin los cuales son ya los binarios para enviar a nuestro fpga.

PASO 3

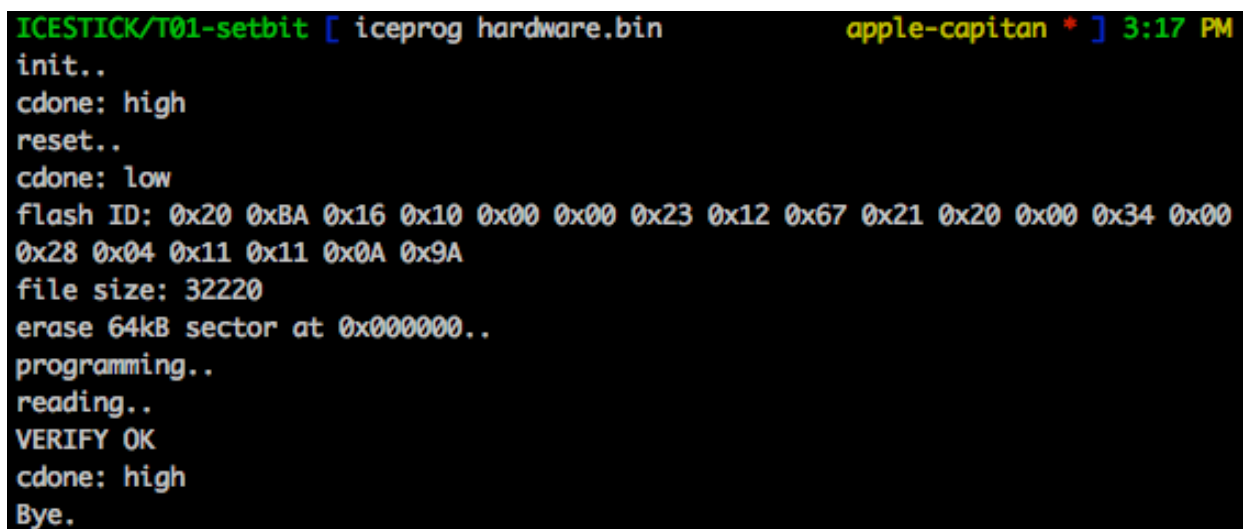
Herramientas libres para la descarga.

En este paso debemos utilizar la herramienta toolchain-icestorm que lo encontramos en <https://github.com/bqlabs/toolchain-icestorm> .

Los pasos para instalar la herramienta esta en el enlace siguiente:

<https://github.com/bqlabs/toolchain-icestorm/wiki#testing-in-mac>

Una vez instalado en nuestro computador podemos ejecutar el archive binario que obtuvimos en el anterior paso y hacemos la descarga al fpga.



```
ICESTICK/T01-setbit [ iceprog hardware.bin apple-capitan * ] 3:17 PM
init..
cdone: high
reset..
cdone: low
flash ID: 0x20 0xBA 0x16 0x10 0x00 0x00 0x23 0x12 0x67 0x21 0x20 0x00 0x34 0x00
0x28 0x04 0x11 0x11 0x0A 0x9A
file size: 32220
erase 64kB sector at 0x000000..
programming..
reading..
VERIFY OK
cdone: high
Bye.
```

utilizamos el comando:

```
$ iceprog code.bin
```

NOTA: Al conectar el USB a la mac debemos cargar el driver del fpga para esto deberiamos primero desconectar el USB primero y ejecutar los siguientes linea de comandos:

```
$ sudo kextunload -b com.FTDI.driver.FTDIUSBSerialDriver
```

```
$ sudo kextunload -b com.apple.driver.AppleUSBFTDI
```

despues de realizer esto conecta el stick y ya podras descargar el archivo.