

## Proyecto de Ingeniería de Computadores (PEC)

### Objetivo del curso

El objetivo de esta asignatura es que el alumno aprenda a desarrollar un prototipo de un computador o un SoC (System on Chip) en un chip programable sobre una placa base para crear un mini-ordenador. Se pondrán en práctica algunos de los conocimientos adquiridos en asignaturas anteriores: diseño de la microarquitectura de un procesador, diseño e implementación de software de sistema, y diseño de sistemas digitales.

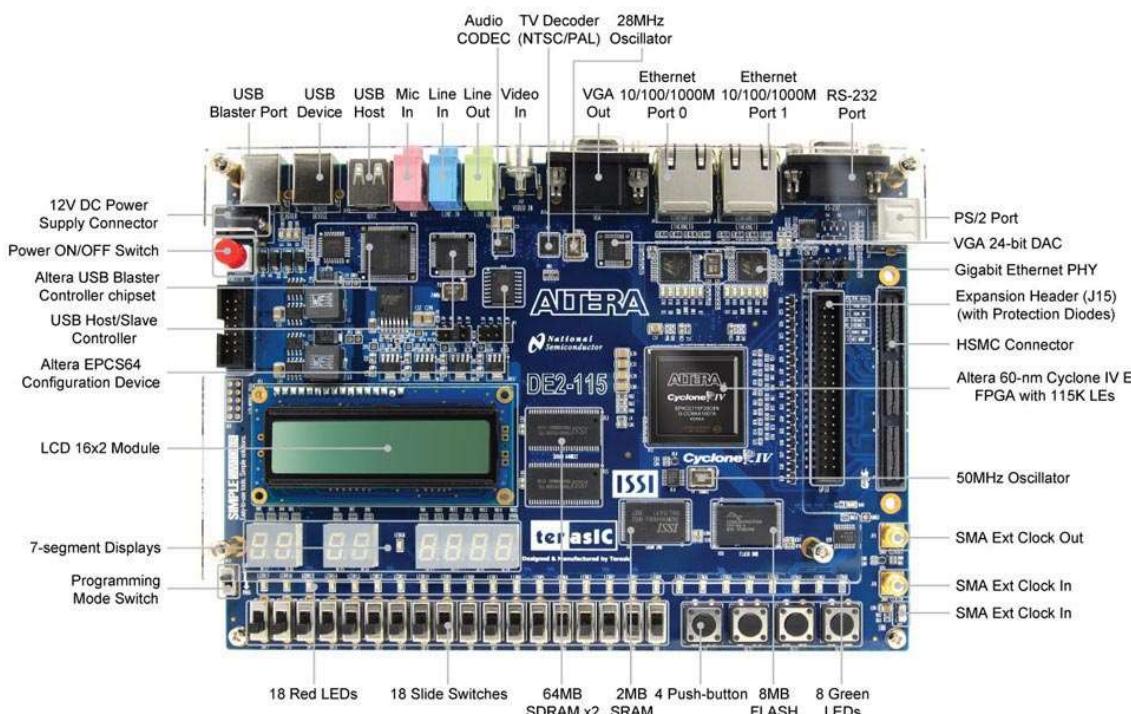
Fases principales del proyecto.

- 1) Aprendizaje de las herramientas de desarrollo para los chips programables (FPGA) y práctica del lenguaje de descripción del hardware VHDL.
- 2) Implementar pequeños componentes o dispositivos en el chip programable de la placa base.
- 3) Implementar una primera versión simplificada del procesador en una FPGA (sin memoria externa, ni soporte para el sistema operativo o dispositivos externos)
- 4) Implementar una versión completa del procesador.
- 5) Programar un sistema de arranque (BIOS) para el Sistema Operativo en el procesador.
- 6) Evaluar el rendimiento de varias aplicaciones sobre la plataforma que se ha diseñado.

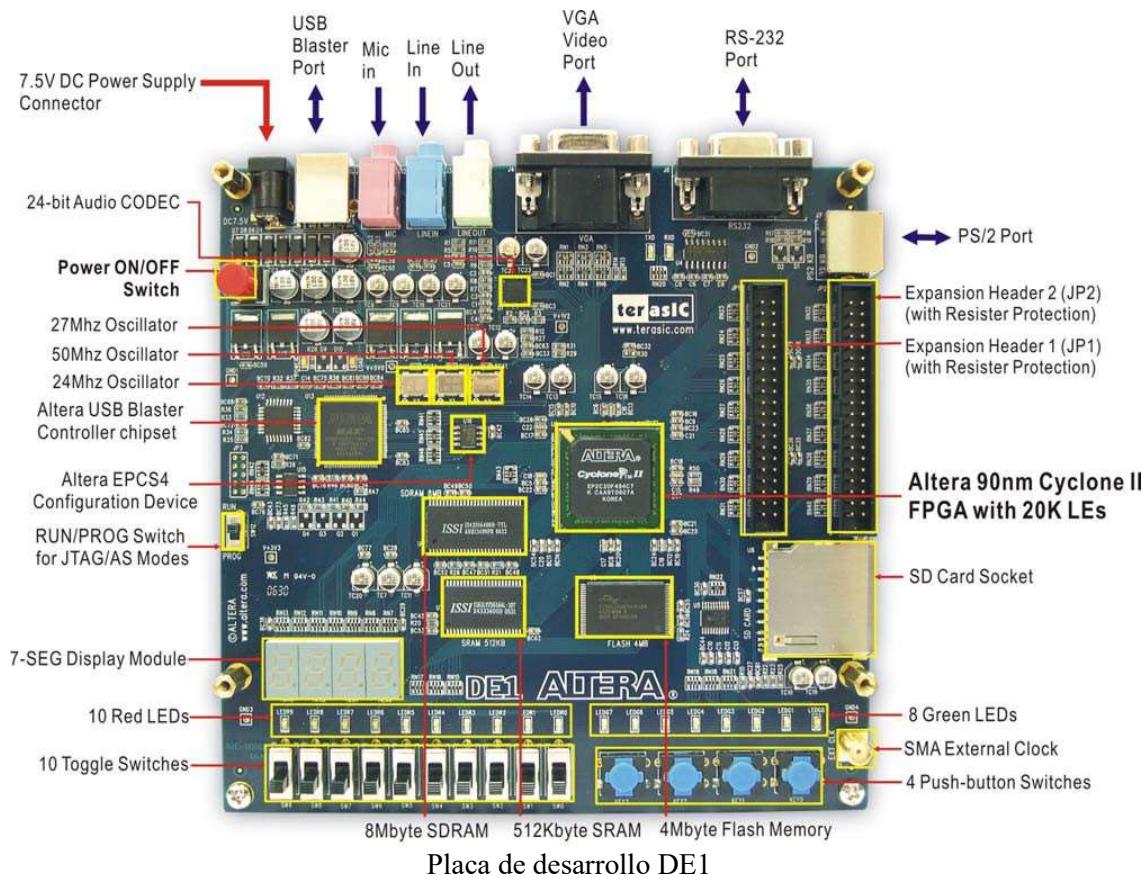
### Objetivo de la sesión

En esta sesión vamos a familiarizarnos con el entorno de desarrollo y con la placa de pruebas. Para ellos diseñaremos diversos circuitos que nos ayudarán a entender el funcionamiento de algunos de los componentes que dispone la placa base. También veremos herramientas y útiles necesarios para ayudarnos a debugar nuestros diseños.

Para la realización de este curso disponemos de dos tipos de placas distintas con una FPGA de la casa *Altera*. Podremos usar indistintamente cualquiera de ellas con sólo algunos pequeños cambios en la configuración de nuestros diseños. Las placas disponibles son la DE1 y la DE2-115, ambas del fabricante *Terasic*. Para nuestros diseños no importará que placa usemos.



Placa de desarrollo DE2-115



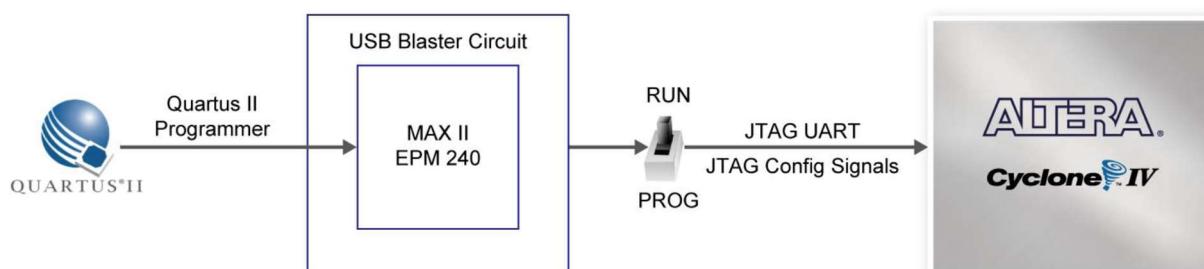
Ambas placas disponen de la potencia y capacidad suficientes, simplemente la placa DE2-115 tiene más capacidad (memoria y unidades programables), más componentes (LCD, Ethernet, ...) y más capacidades de expansión que no usaremos en este curso.

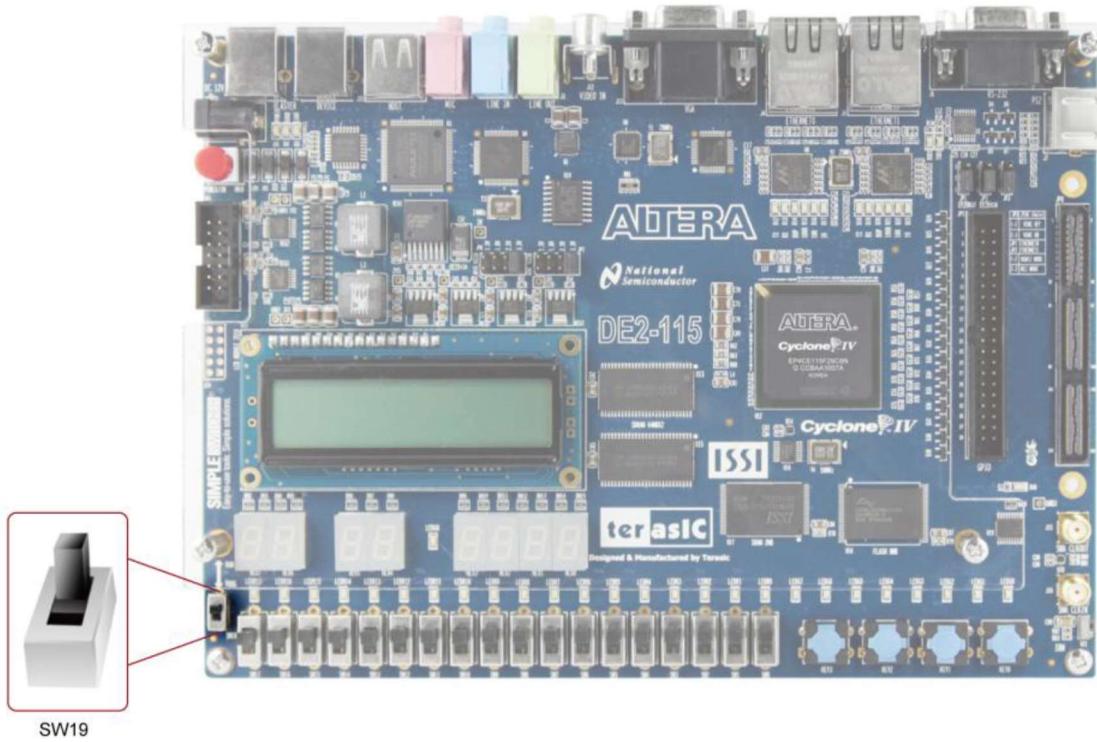
La FPGA que utilizaremos será del modelo *Cyclone* de *Altera* y el software que usaremos para programar estas placas es el *Quartus II Web Edition software*. Es gratuito y se encuentra en la web de Altera.

### Funcionamiento FPGA de la placa de desarrollo

El dispositivo FPGA de la placa se reconfigura cada vez que se enciende la placa y pierde su configuración al desconectarlo de la alimentación. Usando el software adecuado es posible reconfigurar la FPGA tantas veces como queramos. También es posible dejar una configuración almacenada en una EEPROM y que sea esta la que programe la FPGA automáticamente cada vez que se encienda la placa. Al primer método de programación de la placa base se le conoce como JTAG programming (Joint Test Action Group) y al segundo método AS programming (Active Serial).

Durante este curso siempre usaremos el método JTAG. Cada vez que apaguemos la placa perderemos la configuración de la FPGA. Para configurar la placa el modo JTAG simplemente hay que poner el interruptor RUN/PROG en posición de RUN.



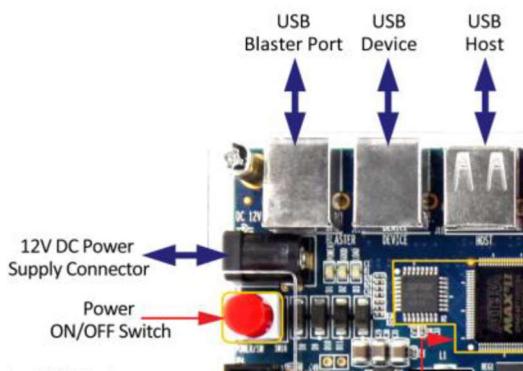


Localización del interruptor RUN/PROG en la DE2-115

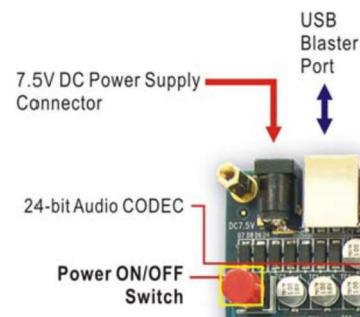
### Conexión de la placa al ordenador

Veamos primero como conectar la placa al ordenador:

- 1) Primero comprobaremos que el interruptor RUN/PROG está en la posición de RUN.
- 2) Enchufamos la placa a la alimentación eléctrica.
- 3) Conectaremos el cable USB a un puerto del ordenador y al puerto *USB Blaster Port* de la placa de desarrollo. Es importante enchufar el conector USB en la ranura más próxima a la alimentación (como se ve en la figura), puesto que los demás puertos sirven para ser configurados y no para grabar diseños en la FPGA.
- 4) Ya podemos pulsar el botón rojo Power ON/OFF Switch, situado al lado de la alimentación.



Conectores DE2-115



Conectores DE1

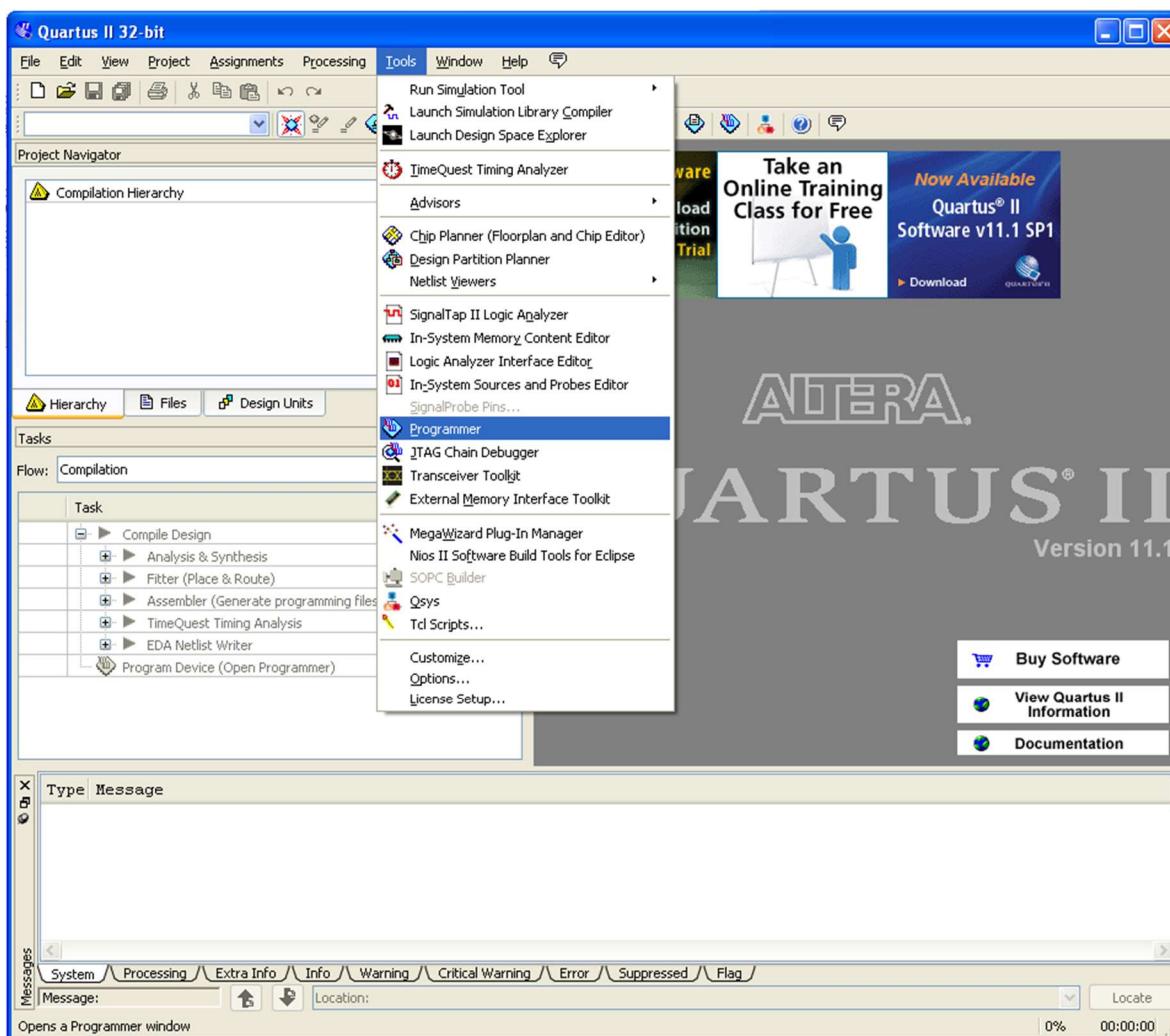
Ahora la placa reprogramará la FPGA con la configuración que esté almacenada en la EEPROM. Si no se ha modificado el programa original veremos que los leds y los 7-segmentos se encienden.

Si es la primera vez que conectamos la placa al ordenador, nos pedirá que instalemos el driver “USB-Blaster driver”. Antes debemos haber instalado el *Quartus II Web Edition software* y en un directorio se encuentra el driver. En el documento “*My First FPGA for Altera DE2-115 Board*” de Terasic podéis ver paso a paso como se instala el driver.

## ¿Cómo programar una FPGA?

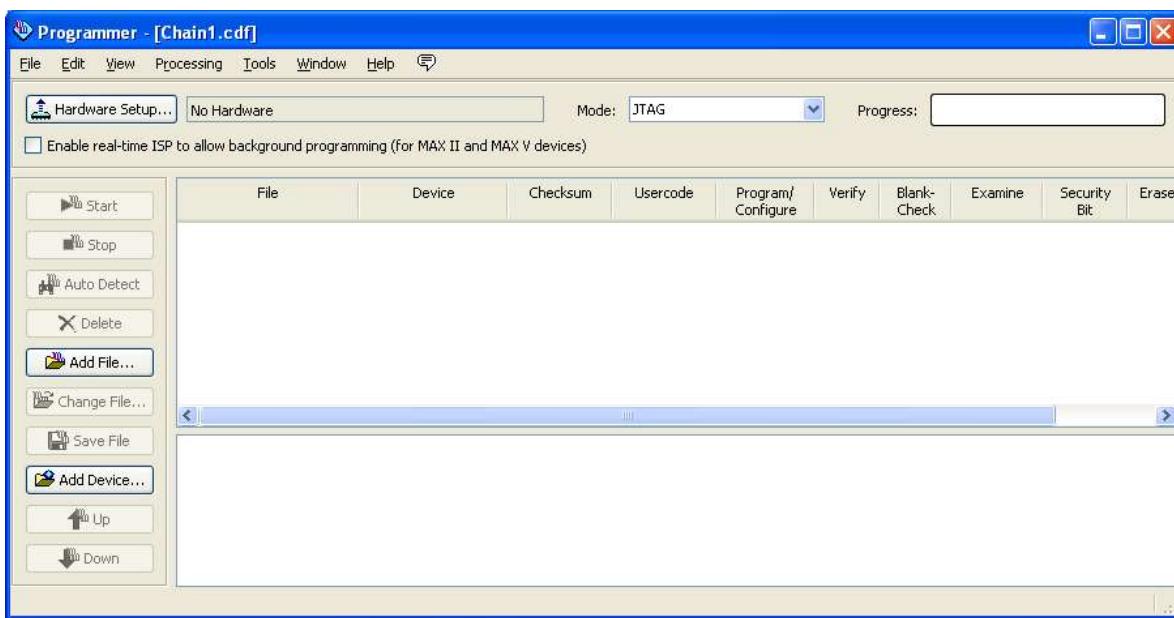
Una vez instalado el software *Quartus II Web Edition software* y el driver “USB-Blaster driver”. Este software permite la creación de diseños para la FPGA (mediante lenguajes de descripción hardware, esquemas, ...) y contiene un conjunto de herramientas. Entre ellas hay la que se encarga de reprogramar las FPGA a través del puerto USB.

Para programar la FPGA primero abrimos la aplicación *Quartus II*. Nos aparecerá una ventana de un asistente con el título “*Getting Started With Quartus II Software*”. La podemos cerrar. Una vez la aplicación esté abierta, vamos directamente a la opción de menú “*tools*” donde están todas las herramientas. Seleccionamos la opción ““*Programmer*” como puede verse en la siguiente figura.

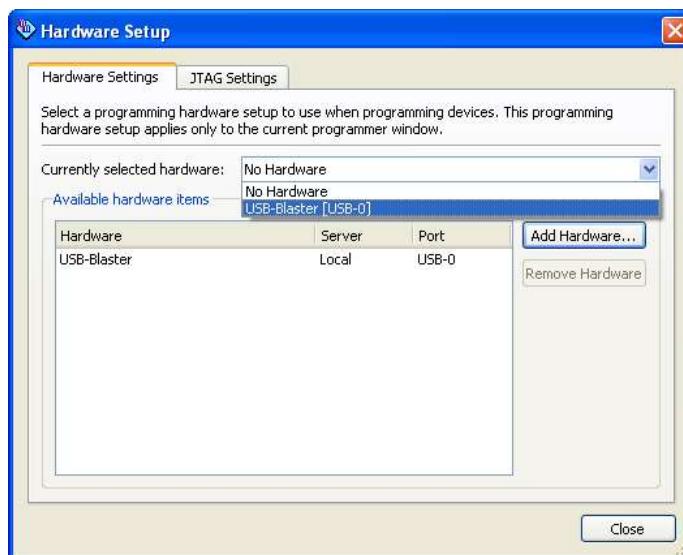


Pantalla principal del Quartus II

Se nos abrirá un nuevo programa, el programa encargado de reprogramar las FPGA. Es el momento de comprobar que la placa está conectada a la alimentación eléctrica y está encendida, que el cable USB está conectado al ordenador y al puerto *USB Blaster*, y que el interruptor RUN/PROG está en la posición de RUN. Si no estaba bien conectada o no estaba encendida, el programador no la detectará.

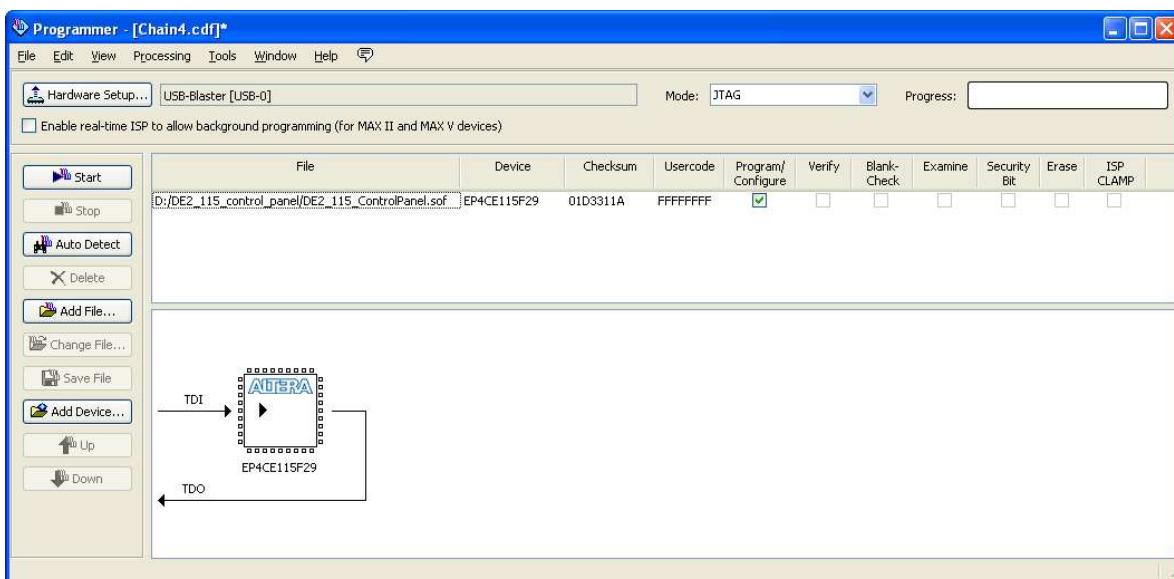
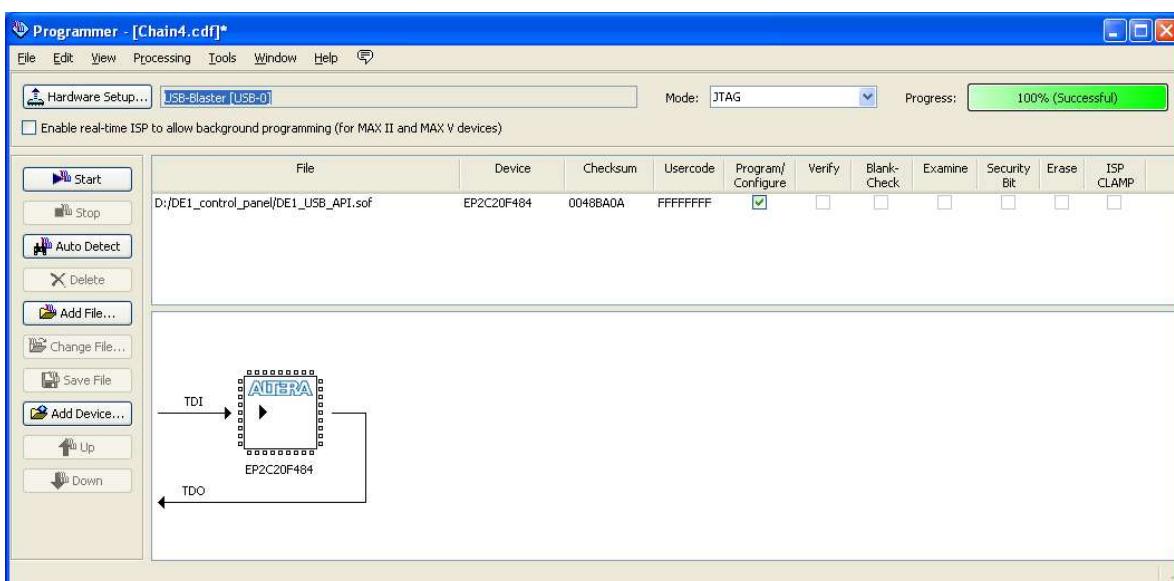


En el caso de que no haya detectado la placa, deberemos configurar el “**Hardware Setup...**” situado en la esquina superior izquierda de la ventana. En el menú desplegable etiquetado como “*Currently selected Hardware:*” debemos seleccionar *USB-Blaster* y pulsar el botón “**Close**”.



A continuación procederemos a seleccionar el programa a cargar. Los programas compilados para reprogramar la FPGA tienen la extensión .sof. Pulsaremos el botón “**Add File...**” y seleccionaremos el archivo con el que deseemos reprogramar la FPGA. Debería aparecer en el recuadro inferior de la ventana un dibujo del chip con el nombre del procesador de nuestra placa. Si todo es correcto, podemos pulsar el botón “**Start**” para comenzar a grabar el programa. Las luces de la placa se apagará durante unos segundos y en la esquina superior izquierda de la ventana aparece una barra de progreso. Cuando esta alcance el 100%, la reprogramación de la FPGA habrá terminado con éxito.

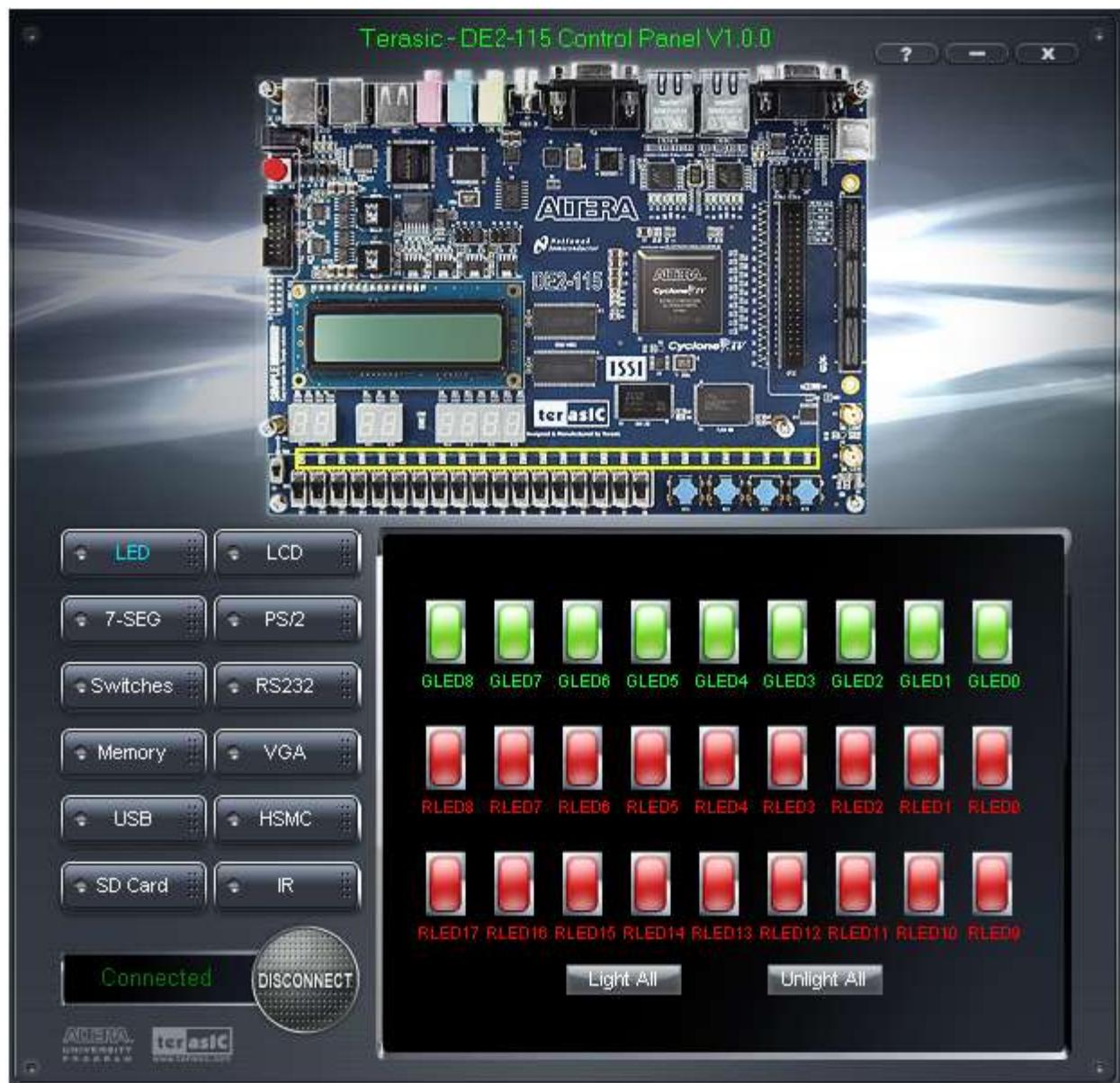
Si estamos utilizando la placa de desarrollo DE2-115 podemos seleccionar como ejemplo el fichero “*DE2\_115\_ControlPanel.sof*”. Entonces veremos como aparece el nombre EP4CE115F29, que es el del procesador de nuestra placa. Para la placa de desarrollo DE1 podemos seleccionar como ejemplo el fichero “*DE1\_USB\_API.sof*” y el nombre que aparecerá es EP2C20F484, que es el del procesador de nuestra placa.

Pantalla del *Programmer* con un fichero para la placa DE2-115Pantalla del *Programmer* con un fichero para la placa DE1

### Herramienta “DE2-115 Control Panel”

Esta herramienta nos servirá para acceder y comprobar el funcionamiento de algunos componentes. Nos permitirá “jugar” con la interfaz de la FPGA (leds, switches, pulsadores, LCD, VGA, ...) y más adelante la usaremos para cargar programas en las memorias (sram, flash, sdram) que lleva incorporadas la placa base. Esta herramienta nos la proporciona el fabricante de la placa de desarrollo ([www.terasic.com](http://www.terasic.com)). Para instalarla simplemente hay que copiar el contenido del directorio “*DE2\_115\_tools/DE2\_115\_control\_panel*” que está en el CD (DE2-115 System CD) suministrado por el fabricante en la carpeta *bin* que está instalado el *Quartus II*.

Para poder usar esta herramienta primero hay que reprogramar la FPGA para que pueda entenderse con ella. Así que abriremos la herramienta de programación de la FPGA (Programmer) y la reprogramaremos con el fichero “*DE2\_115\_ControlPanel.sof*” que está en la misma carpeta que acabamos de copiar. Una vez se haya reprogramado con éxito (la barra de progreso haya llegado al 100%) todos los leds y los 7-segmentos de la placa se encenderán. Ahora ya podemos ejecutar en el ordenador la aplicación del Panel de control. Ejecutamos el archivo “*DE2\_115\_ControlPanel.exe*”, se abrirá una ventana como la de la siguiente figura y automáticamente conectará con la placa.



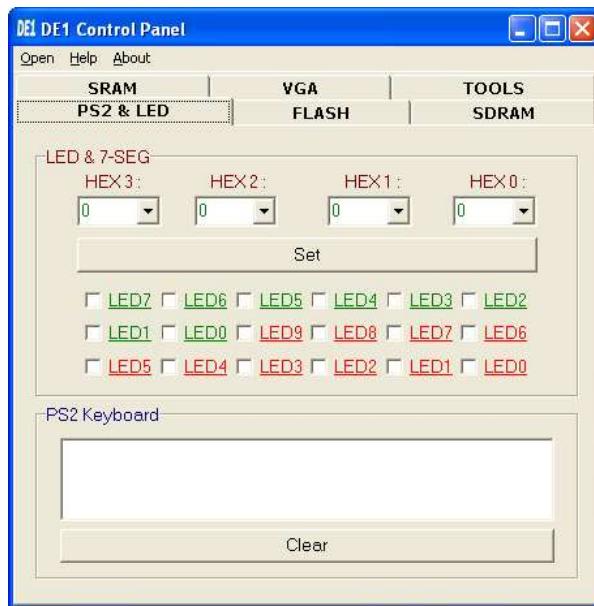
Utiliza la aplicación para probar que todos los dispositivos de la placa funcionan correctamente. Enciende y apaga los leds, activa los *switches*, la pantalla LCD, el sonido...

### Herramienta "DE1 Control Panel"

Esta herramienta nos servirá para acceder y comprobar el funcionamiento de algunos componentes. Nos permitirá “jugar” con la interfaz de la FPGA (leds, *switches*, pulsadores, LCD, VGA, ...) y más adelante la usaremos para cargar programas en las memorias (sram, flash, sdram) que lleva incorporadas la placa base. Esta herramienta nos la proporciona el fabricante de la placa de desarrollo ([www.terasic.com](http://www.terasic.com)). Esta herramienta no hace falta instalarla y se puede ejecutar directamente. Esta herramienta está en el directorio “*DE1\_control\_panel*” que está en el CD suministrado por el fabricante.

Para poder usar esta herramienta primero hay que reprogramar la FPGA para que pueda entenderse con ella. Así que abriremos la herramienta de programación de la FPGA (Programmer) y la reprogramaremos con el fichero “*DE1\_USB\_API.sof*” que está en la misma carpeta. Una vez se haya reprogramado con éxito (la barra de progreso haya llegado al 100%) los visores 7-segmentos de la placa mostrarán el valor “0000”. Ahora ya podemos ejecutar en el

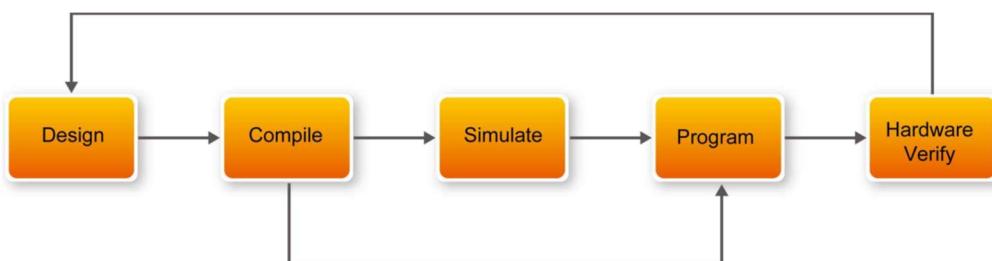
ordenador la aplicación del Panel de control. Ejecutamos el archivo “DE1\_Control\_Panel.exe” y se abrirá una ventana como la de la siguiente figura. Para poder controlar la placa hay ir a la opción “Open” del menú principal y seleccionar la opción “Open USB Port 0”.



Utiliza la aplicación para probar que todos los dispositivos de la placa funcionan correctamente. Enciende y apaga los leds, activa los *switches*, cambia el valor de los visores 7-segmentos...

## Introducción

El diagrama estándar de flujo para la implementación de un circuito lógico usando un dispositivo lógico programable, tal como un chip FPGA (field programmable gate array) sigue las siguientes etapas.



Más concretamente:

En la etapa de diseño (Design) se especifica el circuito deseado, ya sea por medio de un diagrama esquemático, o mediante el uso de un lenguaje de descripción de hardware, como Verilog o VHDL. En esta etapa es donde se crea el circuito digital que se implementará en la FPGA.

En la etapa de compilación (Compile), primero se hace el proceso de Síntesis (Synthesis), el circuito es sintetizado en un circuito formado únicamente por los elementos lógicos (LEs) que dispone el chip FPGA. Luego se hace el proceso de montaje (Fitter). En este proceso la herramienta de compilación determina la ubicación de las LE definidas en la netlist en el chip FPGA (Place), y escoge las rutas de conexión para las señales y buses para hacer las conexiones entre determinados LE (Route).

Habitualmente, en la etapa de compilación también hay un proceso que se encarga del hacer el análisis del tiempo (Timing Analysis). Se analizan y calculan los retardos de propagación a lo largo de los diversos caminos en el circuito implementado en la FPGA para proporcionar una indicación del rendimiento esperado del circuito.

La etapa de simulación (Simulate), aunque es opcional casi siempre es imprescindible. Es la etapa en que se intenta ver que todo funciona correctamente antes de reprogramar la FPGA. Se puede hacer a dos niveles: Se puede hacer una simulación funcional (Functional Simulation) del circuito para verificar el correcto funcionamiento del circuito que va a ser implementado en la FPGA sin tener en cuenta ninguno de los posibles problemas de sincronización, o se puede hacer una simulación de tiempos (Timing Simulation) mucho más precisa para verificar que todas las señales tienen los valores correctos y llegan en el momento adecuado.

La etapa de programación (Program) se encarga de implementar el circuito diseñado físicamente en el chip FPGA mediante la programación de los interruptores de configuración que configuran los elementos lógicos (LEs) y establecer las conexiones del cableado necesario.

La última etapa (Hardware Verify) simplemente se verifica físicamente, ya en el chip FPGA, que todo funciona correctamente.

## Un ejemplo completo paso a paso

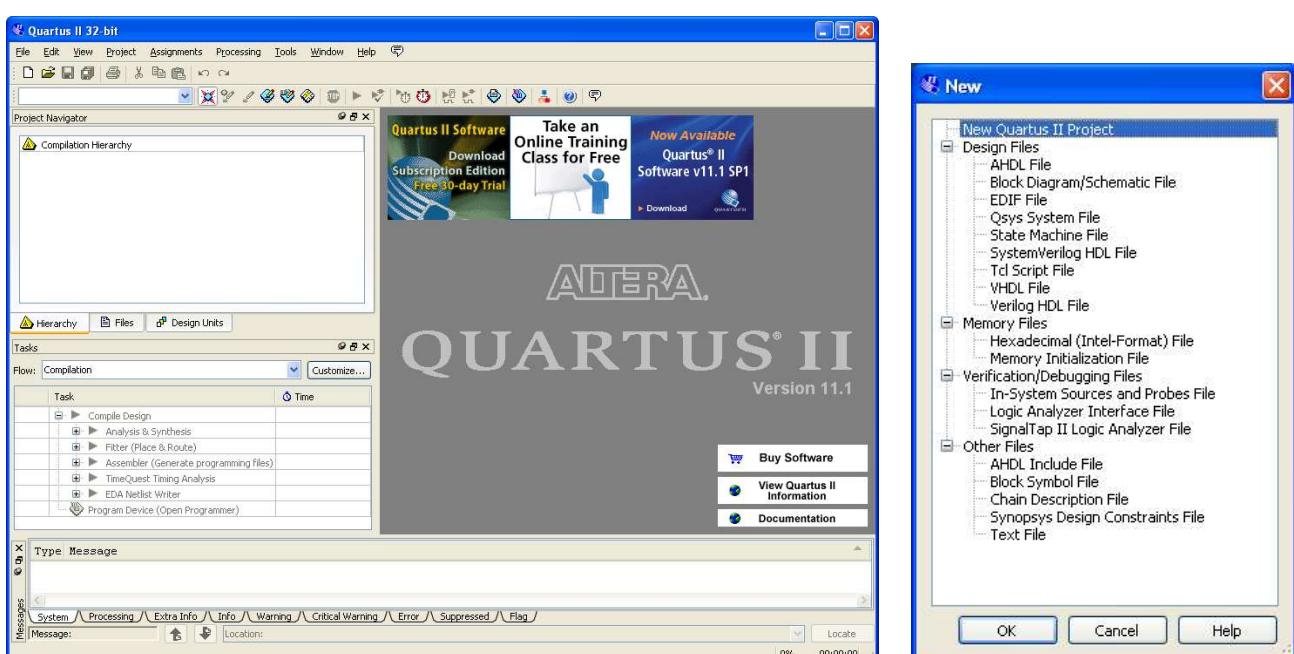
### Diseño esquemático mediante bloques

Para familiarizarnos con todos los programas y procedimientos, primero diseñaremos completamente un sencillo circuito y lo haremos paso a paso usando el software *Quartus II*. Este circuito deberá encender un led de la placa cada vez que se pulsen simultáneamente dos pulsadores de la placa.

El circuito que vamos a crear básicamente contendrá una puerta AND cuyas entradas estarán conectadas a dos pulsadores y su salida a un led.

En primer lugar conecta la placa y enciéndela, tal y como se mostró anteriormente. A continuación ejecuta la aplicación *Quartus II* y una vez iniciada cierra la ventana emergente del asistente.

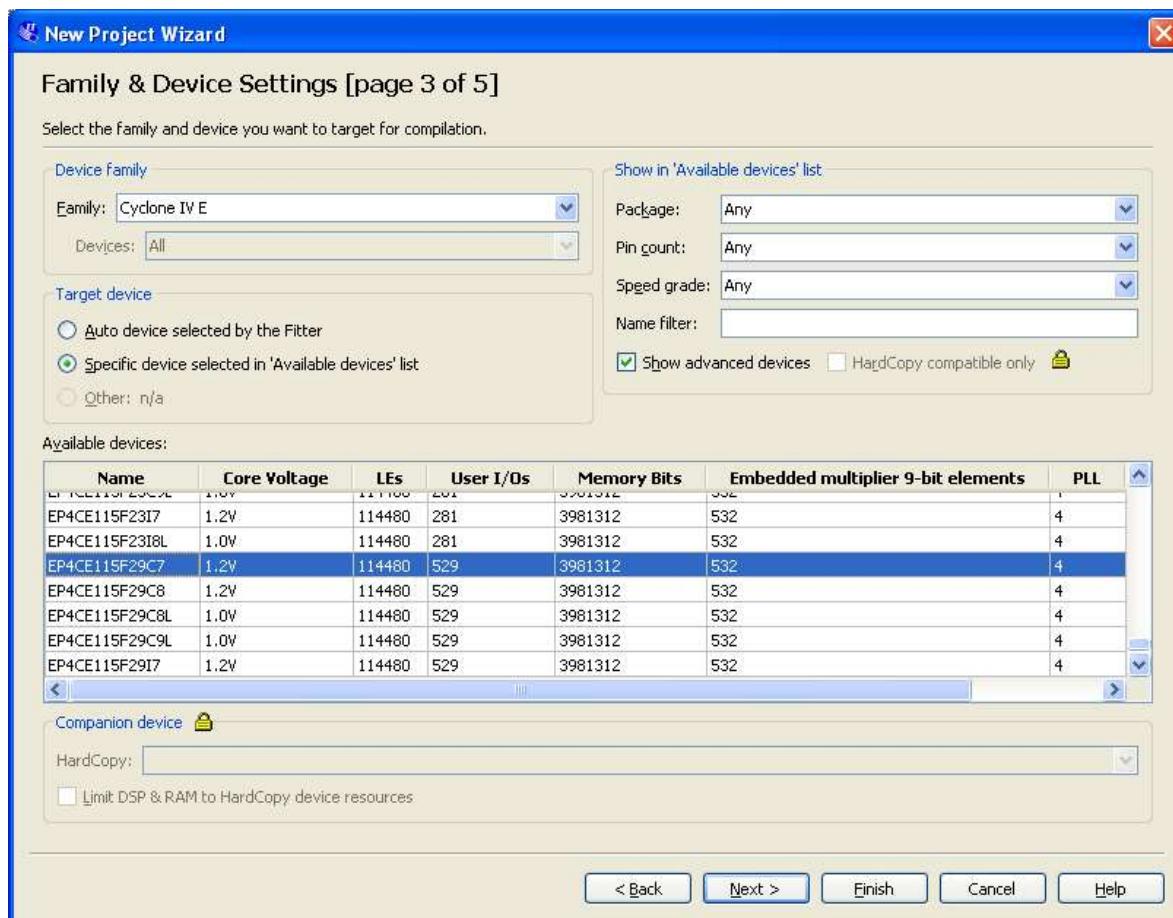
En esta ocasión crearemos un proyecto desde el principio, así que en el menú de pestanas superior selecciona **File→New**. En la ventana que se abre seleccionar “*New Quartus II Project*” y pulsar “OK”.



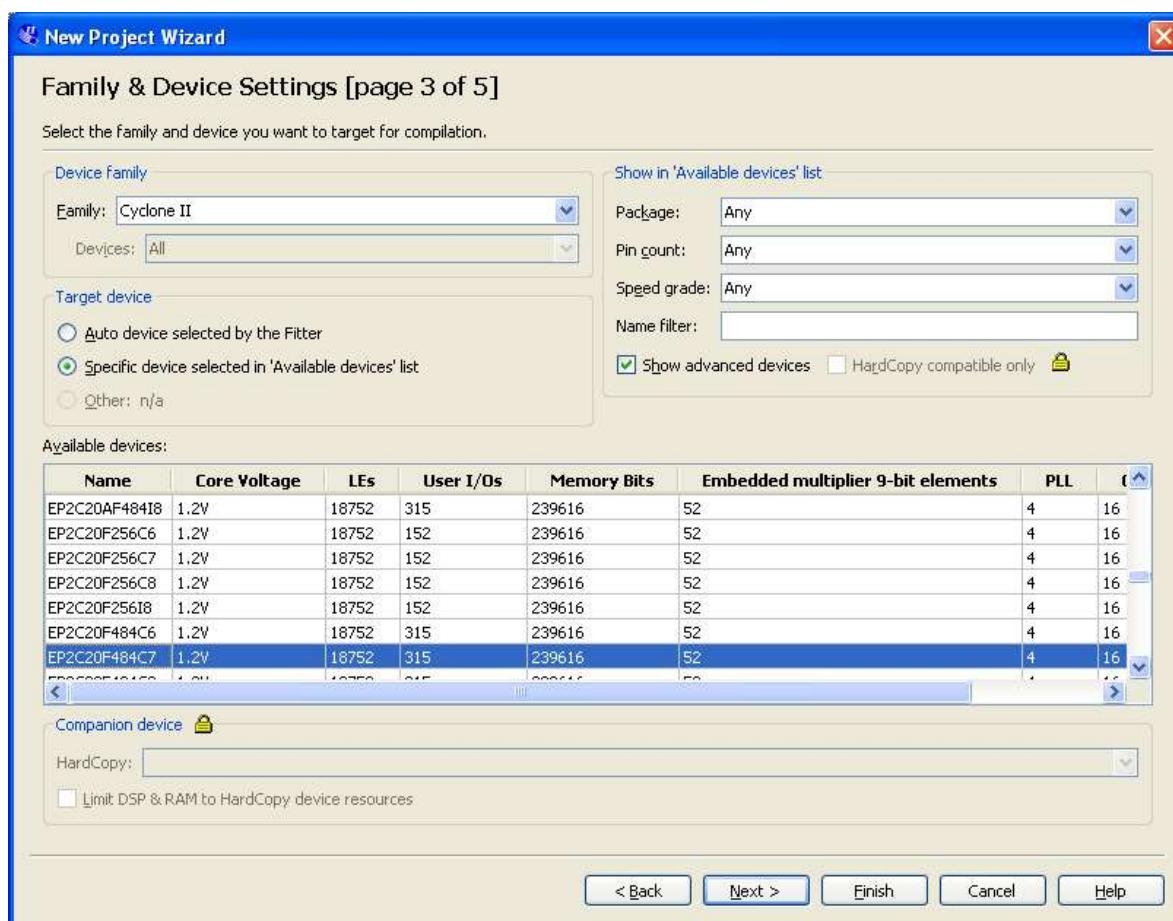
Se abrirá una ventana de título “*New Project Wizard*”. Si no ha sido desactivado, lo primero que muestra esta ventana es una pequeña introducción y en la esquina superior izquierda se leerá el título “*Introduction*”, pulsaremos el botón “*Next*”. En la ventana de directorio, seleccionaremos la carpeta en la que queremos ubicar el proyecto (mejor si está vacía), le daremos un nombre (por ejemplo le llamaremos *DemoAnd2\_sch* al proyecto) y pulsaremos el botón “*Next*”.

La siguiente pantalla, de nombre “*Add Files*” la dejaremos como está y volveremos a pulsar “*Next*”. Llegaremos a una pantalla de nombre “*Family & Device Settings*”. En esta ventana debemos seleccionar la familia y modelo de nuestra FPGA. Este paso es muy importante, puesto que si nos equivocamos el circuito será compilado para otro dispositivo y no podremos grabarlo en la placa.

Si usamos la placa **DE2-115** deberemos seleccionar el dispositivo EP4CE115F29C7 que es el modelo de FPGA que tiene esta placa de desarrollo. En el desplegable etiquetado como “*Family:*” buscaremos **Cyclone IV E**. Esto actualizará el menú “*Available devices:*”, en el que debemos buscar el modelo exacto, **EP4CE115F29C7**. A continuación pulsaremos el botón “*Next*”.



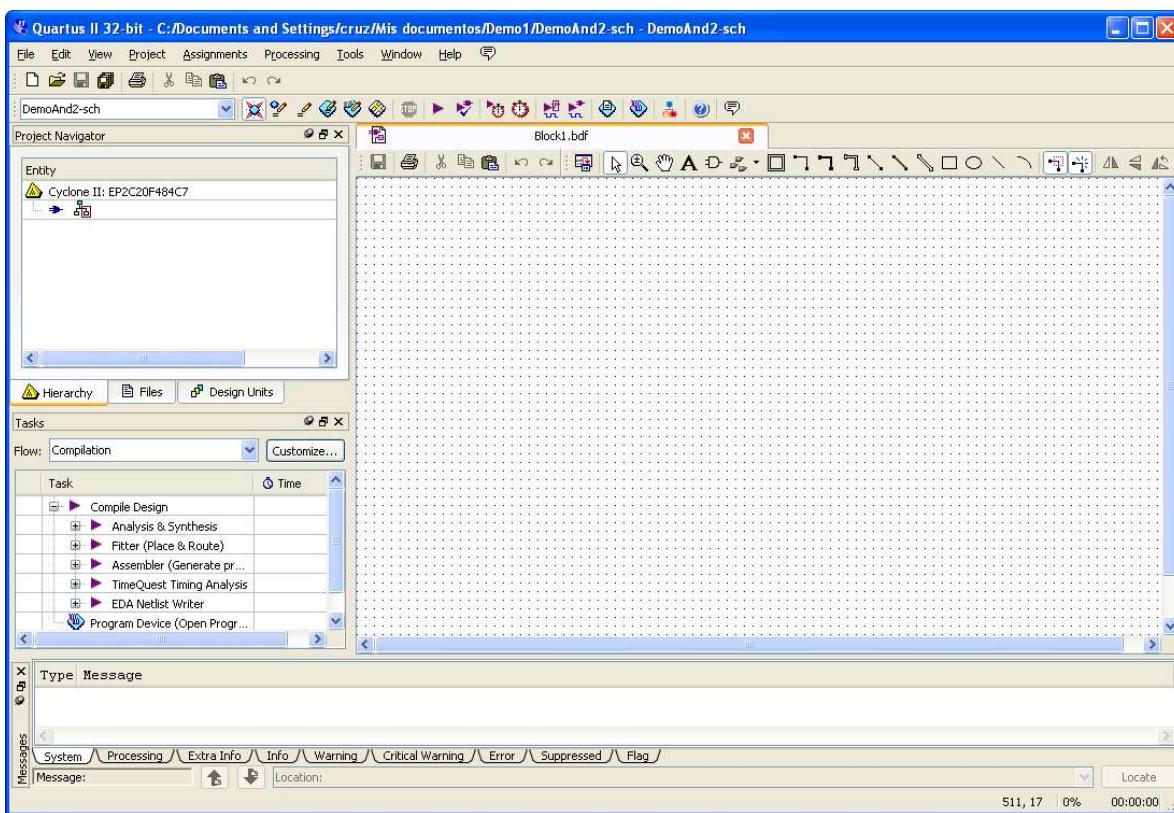
Si usamos la placa **DE1** deberemos seleccionar el dispositivo EP2C20F484C7 que es el modelo de FPGA que tiene esta placa de desarrollo. En el desplegable etiquetado como “*Family:*” buscaremos **Cyclone II**. Esto actualizará el menú “*Available devices:*”, en el que debemos buscar el modelo exacto, **EP2C20F484C7**. A continuación pulsaremos el botón “*Next*”.



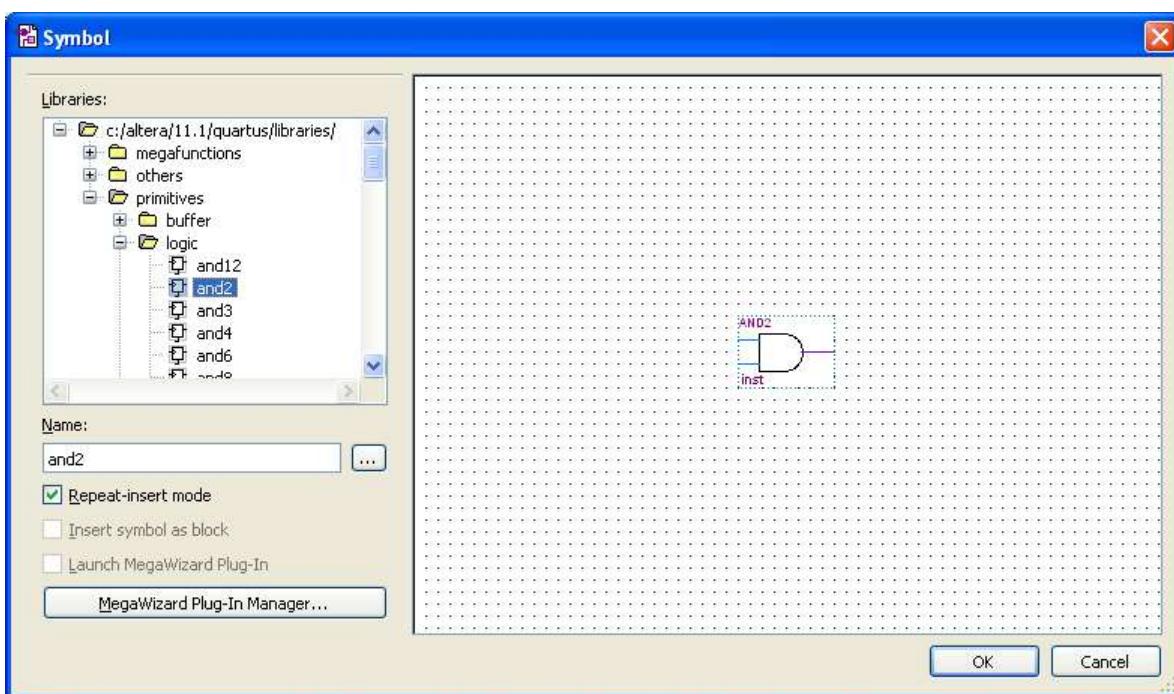
Si alguna vez olvidamos el modelo de FPGA que tiene nuestra placa de desarrollo simplemente hace falta mirar el chip para averiguarlo. En la parte superior de la FPGA está impresa la familia y el modelo exacto.

El siguiente menú lo dejaremos tal y como está, pulsando “Next” otra vez. Finalmente aparecerá un pequeño resumen de las opciones seleccionadas, comprobamos que sean correctas y ya podemos pulsar el botón “Finish”.

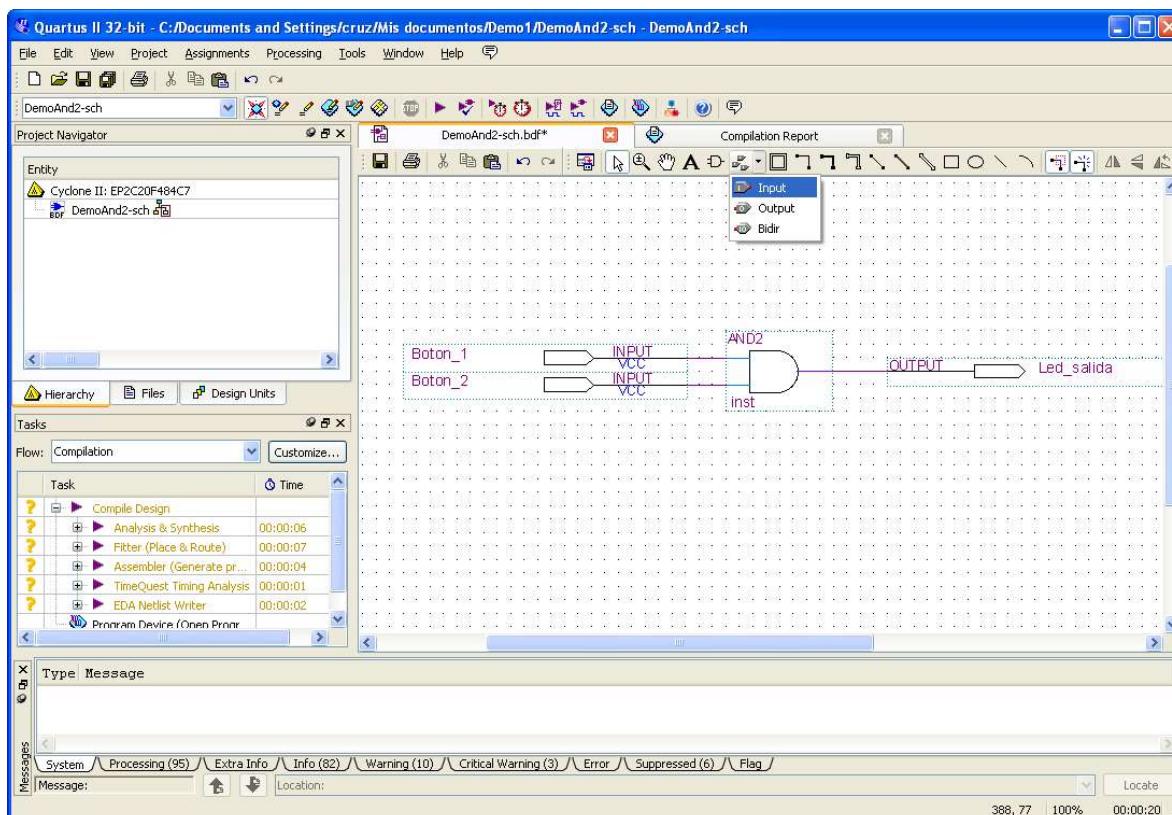
Una vez creado el proyecto, procederemos a crear el diagrama de bloques (Schematic) de nuestro circuito. En el menú de pestañas seleccionaremos **File→New**. Esta vez seleccionaremos la opción “*Block Diagram/Schematic File*” y pulsaremos “OK”. La ventana principal habrá cambiado para mostrar un espacio de trabajo llamado *Block1.bdf* en el que poder diseñar un circuito gráficamente.



En primer lugar insertaremos una puerta AND de dos entradas. Para insertar puertas debemos seleccionar el icono con forma de AND situado en la parte superior del espacio de trabajo. Una ventana de título “Symbol” se abrirá. En el recuadro “Libraries” aparece un menú desplegable en forma de árbol, seleccionaremos c:/altera/11.1/quartus/libraries/→primitives→logic→and2 y pulsaremos “OK”. El cursor cambiará a una forma de cruz con una puerta AND enganchada, que ubicaremos donde creamos conveniente.

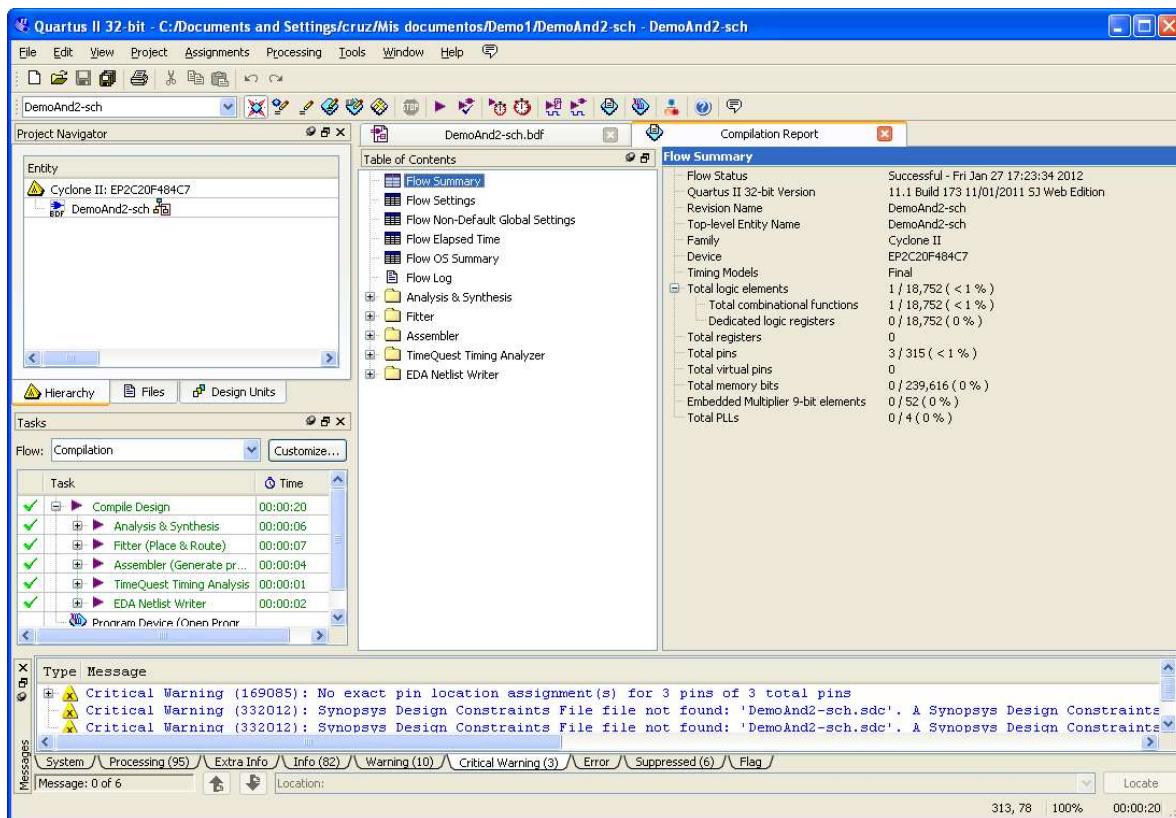


A continuación debemos añadir las entradas y salidas del circuito y conectarlas a la puerta lógica. Para hacer esto pulsaremos la pequeña flecha del ícono (que hay justo a la derecha del ícono en forma de puerta AND) y seleccionaremos “Input”. Colocaremos dos entradas delante de la puerta AND y una salida detrás, seleccionando “Output” en el desplegable anterior. Una vez hecho esto, procederemos a conectar las salidas y la entrada con la puerta AND. Para hacer esto basta con colocar el puntero justo encima de las líneas que representan cables hasta que el puntero adopte la forma de una cruz. Entonces clicamos con el botón izquierdo y aparecerá una línea, que conectaremos en el lugar deseado. Haciendo un “doble click” sobre el nombre de cada uno de los *pin* que acabamos de poner podremos cambiarlos. Renombraremos los dos *pins* de entrada como *Boton\_1* y *Boton\_2*; y el de salida como *Led\_salida*. El circuito debe quedar parecido al mostrado a continuación.



Una vez hecho esto, ya podemos compilar el proyecto. Primero guardaremos el proyecto con **File→Save (Ctrl+S)** y a continuación **Processing→Start Compilation (Ctrl+L)**. Esto iniciará el proceso de compilación que puede tardar unos minutos. Cuando el proceso termine, veremos que hemos obtenido, a parte de una pestaña con el informe de la compilación, varios *warnings*. La mayoría de ellos carecen de importancia y los ignoraremos. Los *warnings* importantes aparecen en la pestaña “*Critical Warning*” situada en la parte inferior de la consola.

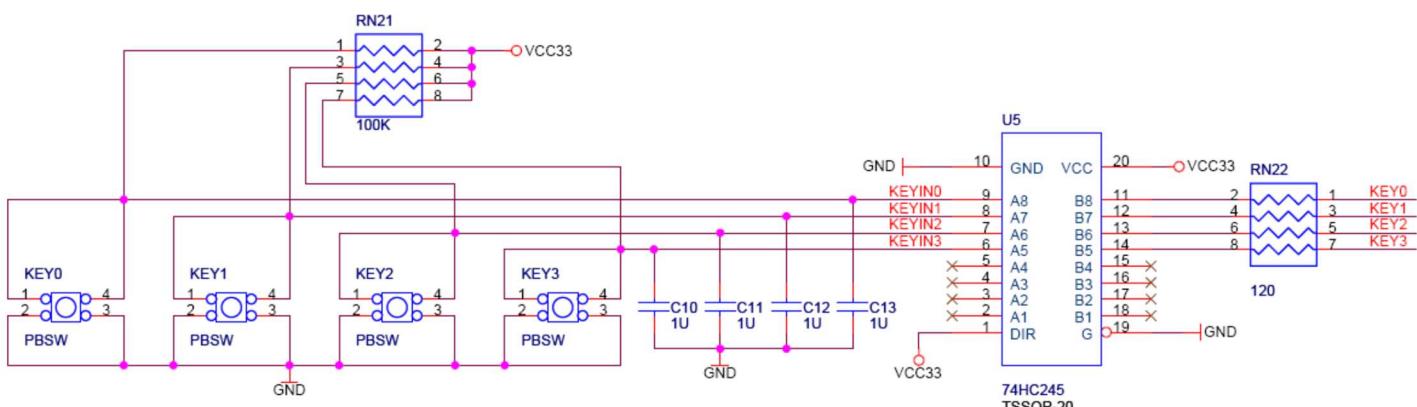
De los avisos que aparecen en “*Critical warning*”, de momento, sólo debe preocuparnos el siguiente: “*Critical Warning (169085): No exact pin location assignment(s) for 3 pins of 3 total pins*”. Este aviso nos informa de que no tenemos asignadas las entradas (inputs) y salidas (outputs) del diseño a ningún *pin* de la FPGA, es decir, no hemos asignado las entradas y salidas a ningún botón ni led de la placa.



Para hacer esta asignación, utilizaremos la herramienta “*Pin planner*”, que se encuentra en **Assignments→Pin Planner**. Esta herramienta permite asignar los nombres de las salidas y entradas que hemos puesto en nuestro diseño a los diversos *pins* (patillas) de la FPGA. Cada uno de los *pins* de la FPGA está conectado a algún dispositivo de la placa de desarrollo. Necesitamos saber a que *pinnes* de la FPGA están conectados los pulsadores y los leds de la placa de desarrollo para poder hacer esta asignación. Esta información nos la proporciona el manual del fabricante de la placa.

Vamos a ver simplemente la parte que nos interesa para este ejemplo. Vemos que ambas placas disponen de 4 pulsadores. El manual nos indicará a que *pinnes* de la FPGA están conectados. Además nos da una información adicional que nos indica que los pulsadores están filtrados contra rebotes cosa que no ocurre con los interruptores (*switch*) de la placa. En la sección 4.2 de los manuales de las placas vienen la descripción, el comportamiento y la asignación de los *pinnes* de los pulsadores.

Podemos ver un esquema de como están conectados a la placa base y lo más importante la tabla de asignaciones.



En esta tabla, por ejemplo podemos ver que el pulsador cero (*KEY[0]*) está conectado al *pin R22* de la FPGA de la placa DE1 (todos los *pinnes* de la FPGA están etiquetados).

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_R22	Pushbutton[0]
KEY[1]	PIN_R21	Pushbutton[1]
KEY[2]	PIN_T22	Pushbutton[2]
KEY[3]	PIN_T21	Pushbutton[3]

Tabla asignaciones placa DE1

Signal Name	FPGA Pin No.	Description
KEY[0]	PIN_M23	Pushbutton[0]
KEY[1]	PIN_M21	Pushbutton[1]
KEY[2]	PIN_N21	Pushbutton[2]
KEY[3]	PIN_R24	Pushbutton[3]

Tabla asignaciones placa DE2-115

En la siguiente tabla podemos ver como están conectados algunos de los leds de la placa base

Signal Name	FPGA Pin No.	Description
LEDG[0]	PIN_U22	LED Green[0]
LEDG[1]	PIN_U21	LED Green[1]
LEDG[2]	PIN_V22	LED Green[2]
LEDG[3]	PIN_V21	LED Green[3]
LEDG[...]	PIN_...	LED Green[...]

Tabla asignaciones placa DE1

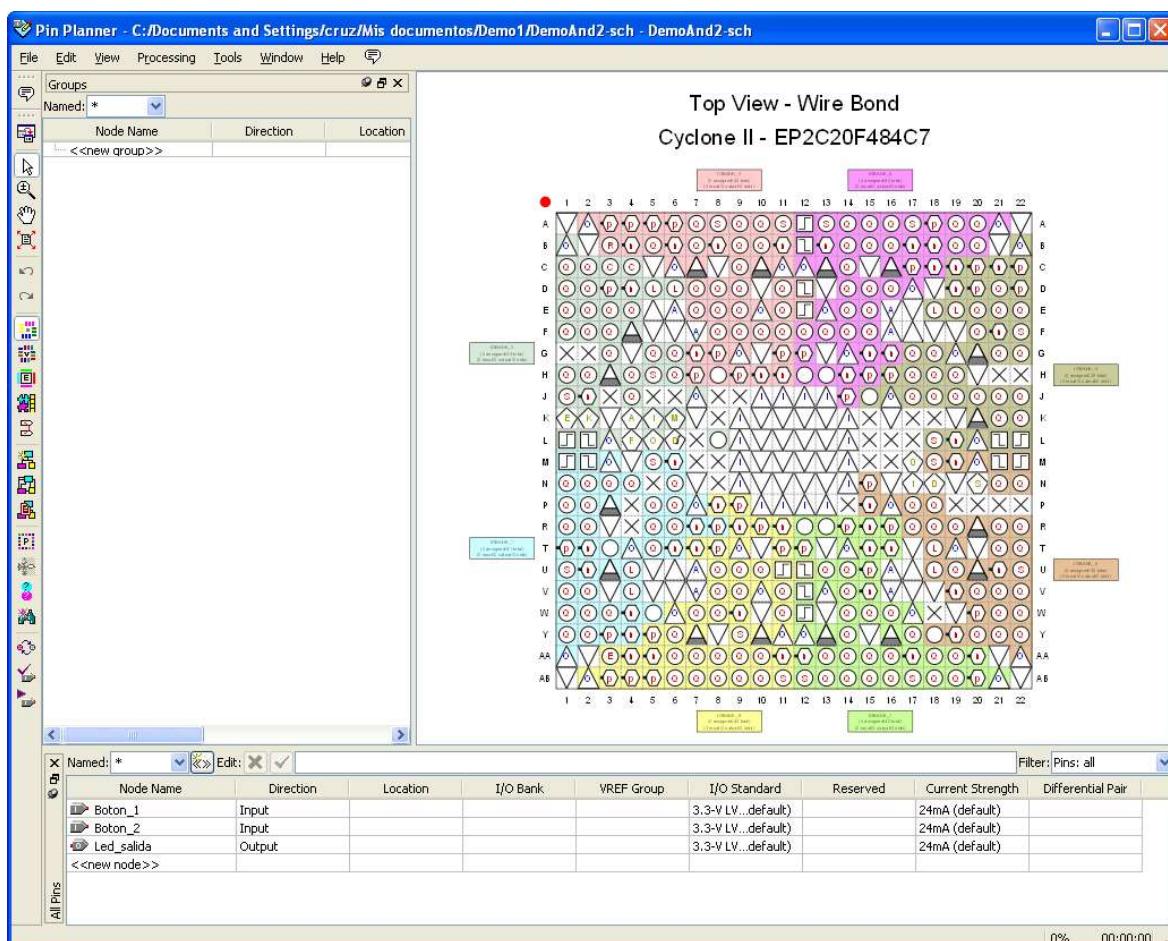
Signal Name	FPGA Pin No.	Description
LEDG[0]	PIN_E21	LED Green[0]
LEDG[1]	PIN_E22	LED Green[1]
LEDG[2]	PIN_E25	LED Green[2]
LEDG[3]	PIN_E24	LED Green[3]
LEDG[...]	PIN_...	LED Green[...]

Tabla asignaciones placa DE2-115

Para asignar las entradas del ejemplo escogeremos dos pulsadores (por ejemplo *KEY[0]* y *KEY[1]*) y un led verde para la salida (por ejemplo el *LEDG[2]*)

En el caso de la placa DE1 serán los *pinnes* R22, R21 y V22, y en el caso de la placa DE2-115 serán los *pinnes* M23, M21 y E25.

Abrimos la herramienta “*Pin planner*”, que se encuentra en **Assignments→Pin Planner**. Nos aparecerá una ventana como la siguiente que nos permitirá hacer la asignación.



Así pues, si usamos la placa DE1, en una de las filas de “*Input*”, escribiremos en la columna “*Location*” R22 y pulsaremos *intro* (nótese que el nombre se completará automáticamente como PIN\_R22), y en la otra fila de “*Input*” pondremos R21. En la fila de *Output* elegiremos el led verde 2, de nombre V22. Una vez hecha las asignaciones deberían quedarnos las filas como en la siguiente figura. En caso de usar la placa DE2-115 hay que utilizar los pinnes M23, M21 y E25 respectivamente.

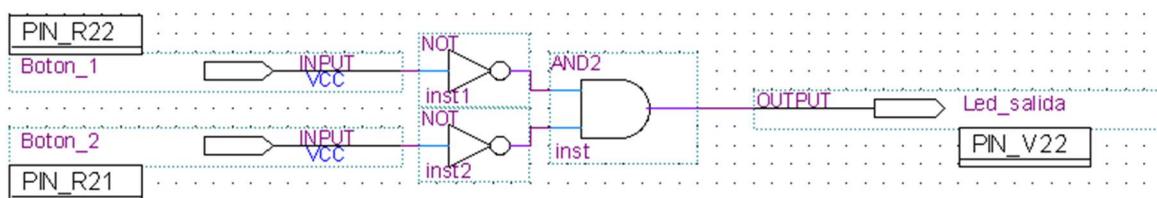
Node Name	Direction	Location
Boton_1	Input	PIN_R22
Boton_2	Input	PIN_R21
Led_salida	Output	PIN_V22
<<new node>>		

Una vez asignados los nombres correctamente seleccionaremos **File→Close** y ya habremos hecho la asignación.

Ahora podemos volver a compilar del mismo modo que antes, **Processing→Start Compilation** y veremos como esta vez no tenemos los *warnings* referentes a los pins. Una vez compilado el circuito, ya podemos proceder a grabarlo en la placa de la misma forma que se detallaba en un apartado anterior de este manual.

Básicamente hay que seleccionar **Tools→Programmer**. Comprobar que en “*Hardware Setup...*” aparece USB-Blaster como opción seleccionada, de no ser así seguir los pasos explicados en el anterior apartado. En el recuadro central debería aparecer ya el fichero con el nombre del proyecto que hemos elegido y extensión “*.sof*”. En nuestro caso *DemoAnd2\_sch.sof*. Así pues, ya podemos pulsar “*Start*” para comenzar a grabar el circuito.

Cuando la grabación haya terminado, veremos cómo se enciende un led verde. Este se apagará al pulsar cualquiera de los dos pulsadores configurados. Aparentemente no funciona como esperábamos, no hace la AND entre los dos botones. Esto es debido a que (como claramente se indica en los manuales de la placa) un pulsador no pulsado envía un 1, mientras que un pulsador pulsado envía un 0, por lo que la luz se apagará al pulsar cualquiera de los dos pulsadores, y permanecerá encendida sólo si ninguno de ellos está pulsado. Para arreglarlo y que haga lo que nos habíamos propuesto, simplemente hay que añadir una puerta lógica NOT entre cada una de las entradas del circuito y su correspondiente entrada de la puerta lógica AND. Tal y como muestra la siguiente figura.



Ahora simplemente volvemos a compilar el proyecto y a grabarlo en la FPGA. Ya podemos comprobar que funciona correctamente.

### Diseño mediante el lenguaje de descripción hardware VHDL

Durante este curso raramente vamos a diseñar algún circuito a nivel de puertas. Utilizaremos el lenguaje VHDL para la descripción del hardware.

En este apartado crearemos de nuevo el mismo circuito que en el anterior, pero esta vez utilizando VHDL. También veremos cómo ahorrarnos el engorroso proceso de asignación de *pinnes* utilizando nombres estándar *Altera*, los que figuran en los manuales, y como visualizar un esquema de bloques utilizando la herramienta “*RTL Viewer*”.

Con el proyecto ya creado (al proyecto le llamaremos *DemoAnd2\_vhdl*), selecciona **File→New** en el menú de pestañas superior. En la ventana emergente selecciona **Design Files→VHDL file**. En el fichero que se abre crearemos el diseño de una AND utilizando el lenguaje de definición de hardware VHDL.

A continuación se muestra el código:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY DemoAnd2_vhdl IS
    PORT( KEY      : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
          LEDG     : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END DemoAnd2_vhdl;

ARCHITECTURE Structure OF DemoAnd2_vhdl IS
BEGIN
    LEDG(2) <= not(KEY(0)) and not(KEY(1));
END Structure;

```

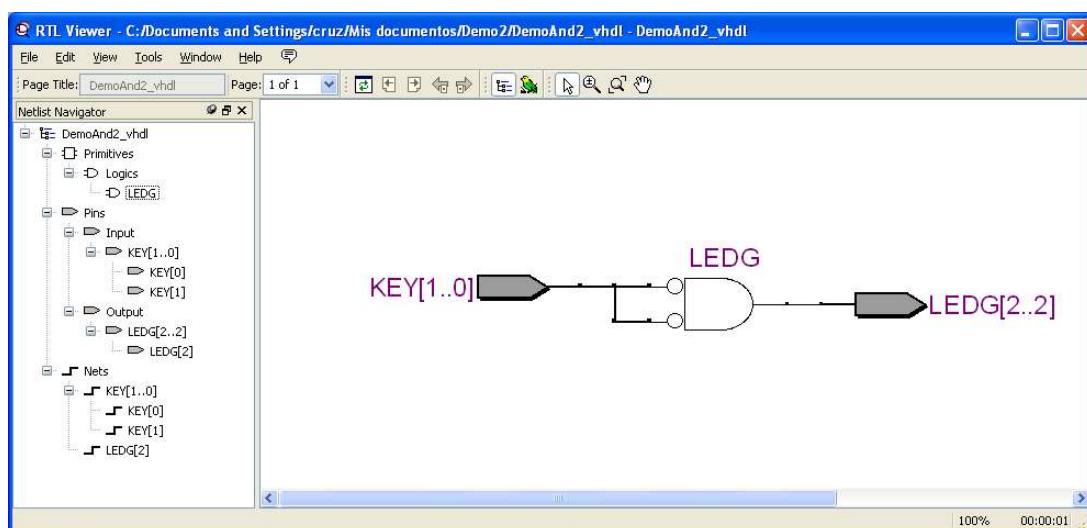
Es importante observar como los nombres utilizados para definir los puertos de entrada (*KEY*) y salida (*LEDG*) coinciden con los nombres estándar de los puertos correspondientes a los botones y al led verde que utilizamos en el apartado anterior. Utilizando estos nombres y realizando una importación podremos ahorrarnos la asignación manual de los *pinnes*.

Antes de terminar con el código, es muy importante que nos fijemos que el nombre de la *ENTITY* coincide con el nombre del proyecto (en nuestro ejemplo *DemoAnd2\_vhdl*). Esto se debe a que esta entidad será identificada como *top-level* y esta debe coincidir con el nombre del proyecto para poder ser compilada. Deberéis tener esto en cuenta en vuestros futuros diseños.

Ahora ya podéis salvar el fichero y guardararlo con el nombre del proyecto. Para evitar la asignación de los *pinnes* manual aparte de usar nombres estándar debemos importar un archivo de asignación. Seleccionad **Assignments**→**Import Assignments...** y busca el fichero “*DE1\_pin\_assignments.csv*” si estáis utilizando la placa DE1 o el fichero “*DE2\_115\_pin\_assignments.qsf*” si estáis utilizando la placa DE2-115.

Por último, antes de compilar es necesario seleccionar la entidad *top-level*. En el recuadro “*Project Navigator*”, situado en la parte superior izquierda de la pantalla, seleccionad la pestaña “*Files*”. En esta ventana debería aparecer un archivo .vhd con el nombre que le habéis asignado al proyecto. Pulsad con el botón derecho del ratón sobre él y seleccionad “*Set as Top-Level Entity*”. Ahora ya podéis compilar, grabar y comprobar que el diseño funciona en la placa.

*Quartus II* también proporciona una herramienta llamada “*RTL Viewer*” muy útil para ver el diseño implementado mediante un esquema de bloques. Seleccionar **Tools**→**Netlist Viewers**→**RTL Viewer**. Se abrirá una ventana en la que podréis observar la AND y las NOT que habéis diseñado conectadas a los botones y al led verde.



## Tareas a realizar

Ahora vais a diseñar e implementar en la placa un conjunto de circuitos que os permitirán recordar los conceptos de VHDL y aprender a usar algunos de los componentes y herramientas que serán muy útiles a la hora de implementar un procesador.

### Tarea 1

Cread un proyecto en VHDL en el que asignéis los *switches* a los leds rojos, de tal manera que el switch[i] encienda y apague el LEDR[i] y comprobad que funciona. Cuando el switch[i] esté a 1 su *led* asociado deberá estar encendido y cuando esté a 0 el *led* deberá estar apagado.

Recuerda al crear el proyecto que debes seleccionar el chip Cyclone II **EP2C20F484C7** como destino si estás usando la placa **DE1** o el chip Cyclone IV E **EP4CE115F29C7** si estas usando la placa **DE2-115**.

Si necesitáis ayuda con la sintaxis de VHDL, podéis recurrir a la opción “*Insert Template*” de *Quartus II*. Una vez estéis editando un fichero, esta opción os aparecerá como un ícono en la barra de iconos de la ventana.

A continuación se muestra el código que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

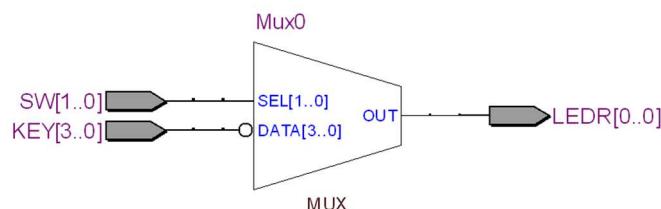
ENTITY Tarea1 IS
    PORT( SW      : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
          LEDR   : OUT STD_LOGIC_VECTOR(9 DOWNTO 0));
END Tarea1;

ARCHITECTURE Structure OF Tarea1 IS
BEGIN
    .
    .
    .
    . -- Aquí debéis poner vuestro código que implemente la tarea
    .
    .
END Structure;
```

### Tarea 2

Cread otro proyecto en VHDL en el que implementaremos un multiplexor 4-1 de un bit. Utilizaremos dos *switch* como señales de control del multiplexor y los 4 pulsadores como señales de entrada. Finalmente conectaremos la salida del multiplexor a un *led* (a vuestra elección).

De modo que con los dos *switch* podremos controlar que pulsador seleccionamos. Este será el pulsador que activará el *led*. Sólo cuando pulsemos el botón el *led* deberá encenderse.



Hay diversas formas de implementar este circuito en vhdl. Recordad que si usáis una estructura formada por las sentencias IF, THEN, ELSIF y ENDIF, estas sentencias son secuenciales. En cambio, si usáis las sentencias WHEN/ELSE y WITH/SELECT para implementar el multiplexor (opción recomendada), estas sentencias son concurrentes.

A continuación se muestra el código que debéis completar. Este código usa el *led* rojo 0 y los *switch* 0 y 1.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Tarea2 IS
    PORT( SW    :  IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
          KEY   :  IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
          LEDR  :  OUT STD_LOGIC_VECTOR(0 DOWNTO 0));
END Tarea2;

ARCHITECTURE Structure OF Tarea2 IS
BEGIN
    .
    . -- Aquí debéis poner vuestro código que implemente la tarea
    .
END Structure;

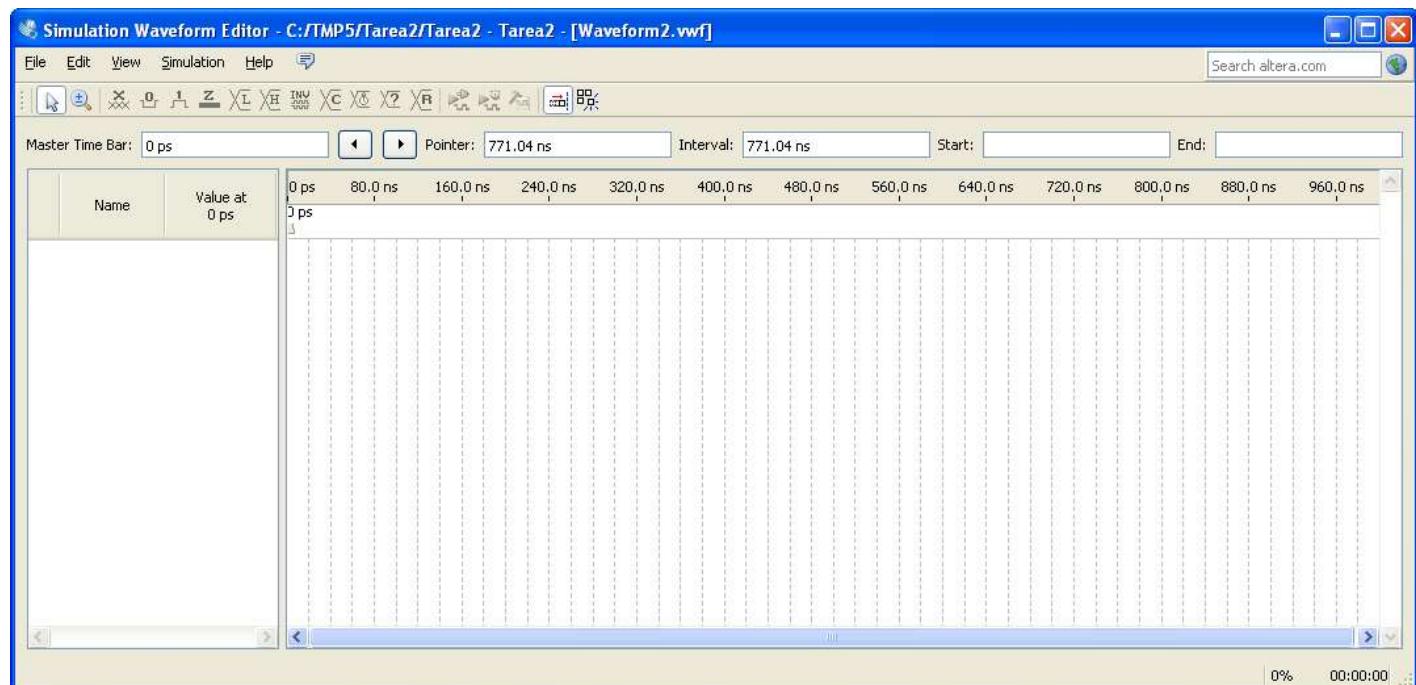
```

### Tarea 3

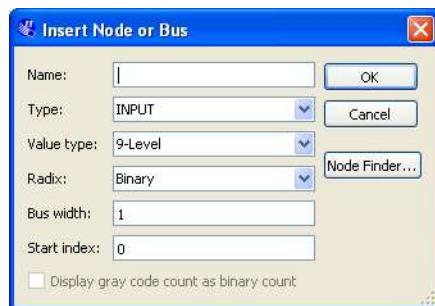
A medida que la complejidad de los diseños se incrementa puede ser muy útil comprobar y verificar el funcionamiento de partes sueltas para detectar errores de funcionamiento. Para ello disponemos de dos simuladores de circuitos. El primero se llama “*ModelSim-Altera 10.1d Starter Edition*”. Es un simulador muy completo y potente pero también difícil de manejar. El otro simulador es el *Qsim* de la propia Altera. Este simulador es mucho más sencillo y orientado al desarrollo de los primeros proyectos.

Con el diseño compilado del multiplexor 4-1 anterior, vamos a usar el simulador *Qsim* para ver las señales y así comprobar que nuestro diseño funciona correctamente.

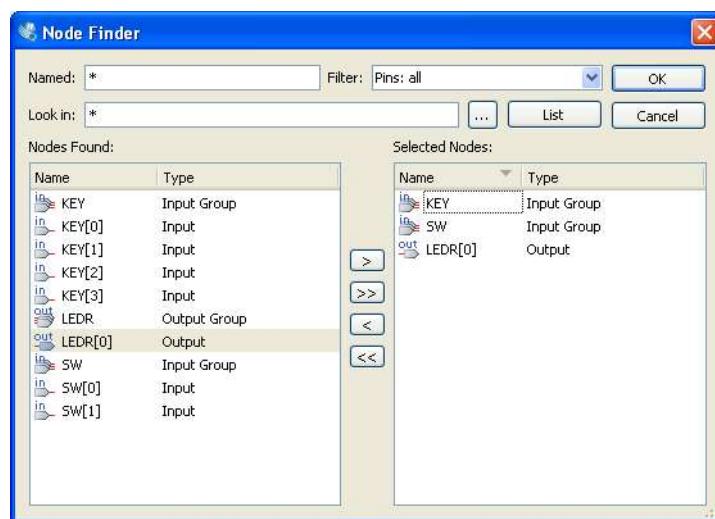
Vamos a crear un cronograma de prueba para nuestro diseño y lo añadiremos al proyecto, así que en el menú de pestañas superior seleccionad **File→New**. En la ventana que se abre seleccionad “*University Program VWF*” en el apartado “*Verification/Debugging Files*” y pulsar “**OK**”. Se os abrirá una nueva ventana similar a la de la figura siguiente con el editor “*Simulation Waveform Editor*”.



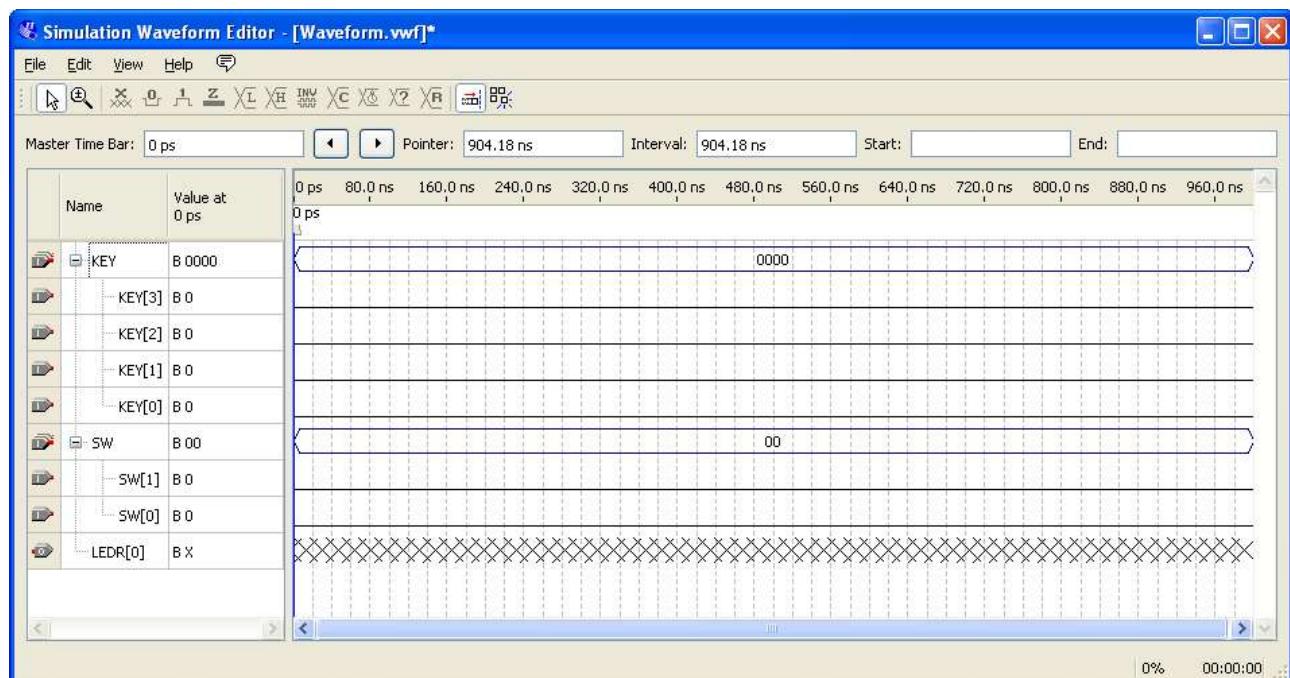
Vamos a proceder a seleccionar las señales que deseamos simular. Seleccionamos en el menú la opción **Edit→Insert→Insert Node or Bus....** En la ventana que nos aparece seleccionamos “**Node Finder ...**”.



En la siguiente ventana pulsamos en “List” y nos aparecerán todas las señales de nuestro proyecto. Mediante los botones “>” y “>>” pasamos a la lista de la derecha aquellas señales que nos interesen en el orden que deseamos. Seleccionamos el grupo de señales *KEY* y *SW* y la señal *LEDR[0]*.

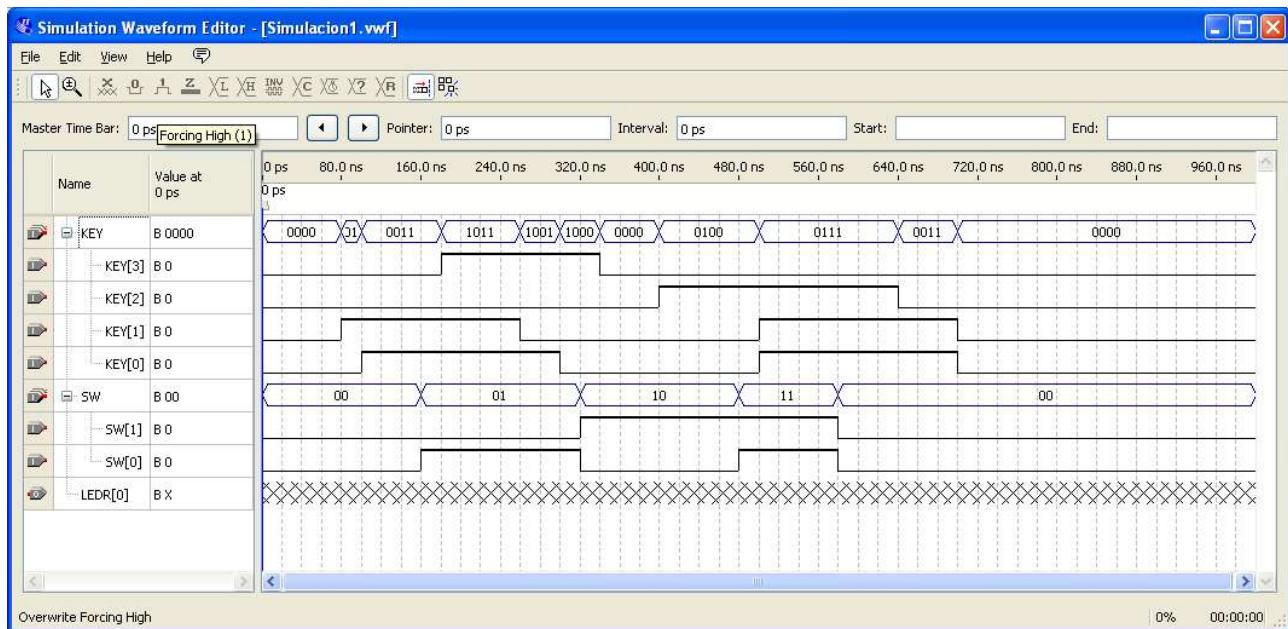


Una vez seleccionadas las señales, le damos a “OK” para cerrar la ventana del “Node Finder” y le volvemos a dar otra vez al “OK” en la ventana de “Insert Node or Bus”. De modo que nos quedará el simulador como la siguiente figura. Por defecto todas las señales de entrada valdrán 0 y las salidas no tendrán valor hasta que las simulemos.



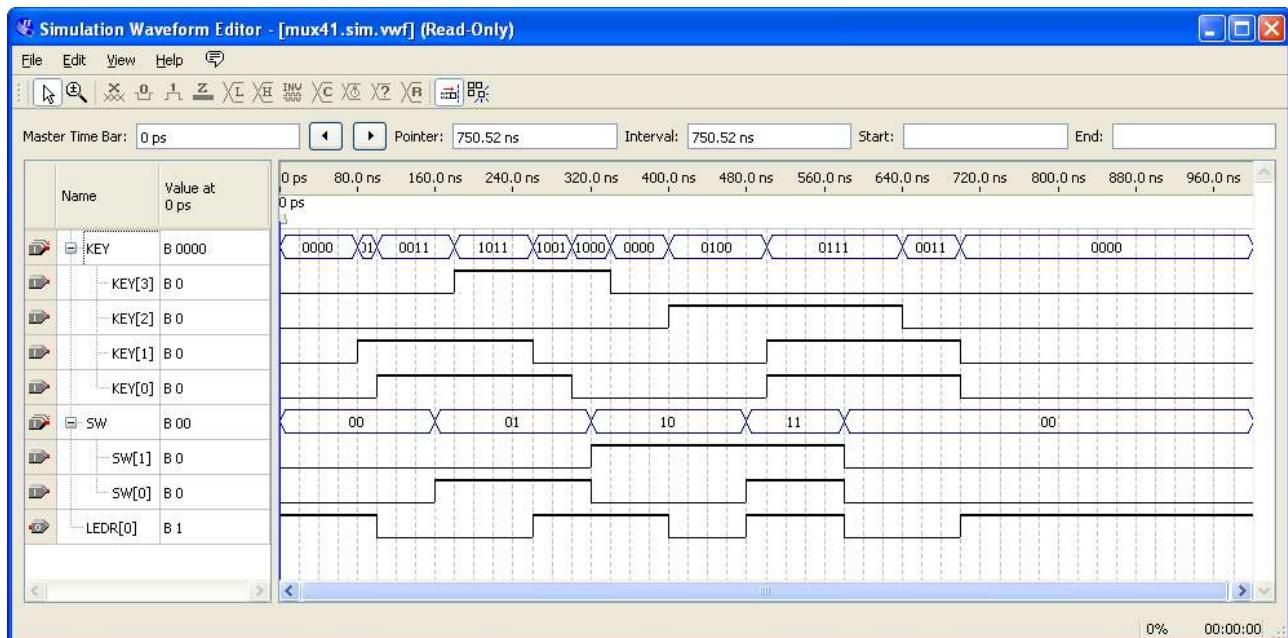
Ahora ya podemos editar el *waveform* para las señales de entrada como queramos. Para ello podemos desplegar las señales que están agrupadas. Con el ratón, mientras mantenemos pulsado el botón izquierdo, seleccionamos sobre una

señal (o varias) el intervalo de tiempo al que queremos asignarle un valor. Una vez seleccionado el intervalo con los iconos del menú, seleccionamos el valor que deseamos que tenga. Después de varias modificaciones podría quedar similar a la siguiente figura.



Finalmente guardamos el cronograma de prueba con la opción **File→Save** y quedará incorporado en nuestro proyecto para poder usarlo tantas veces como queramos. Podemos crear tantas simulaciones distintas como queramos y almacenarlas en nuestro proyecto.

Ahora vamos a simular el comportamiento de nuestro proyecto. Antes de todo comprobad que tenéis seleccionado el simulador “*Quartus II Simulator*”. Para ello id a la opción **Simulation→Options**. Para ver el comportamiento de nuestro circuito haremos una simulación funcional (sin tener en cuenta los tiempos de propagación de las señales). Seleccionamos la opción **Simulation→ Run Functional Simulation** y finalmente se nos abrirá una nueva ventana, como la de la siguiente figura, con el resultado de la simulación.

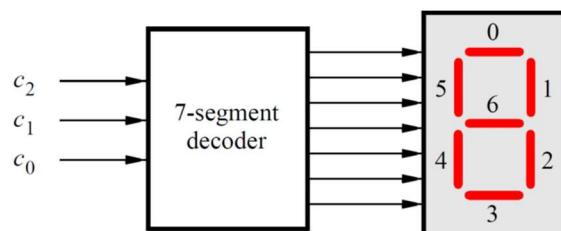


También podemos hacer una simulación que tenga en cuenta los tiempos de propagación de las señales (estos tiempos dependen de diversos factores como del tipo de FPGA que usemos, el *routing* que ha decidido el programa de compilación, etc.). Para ello simplemente tendremos que seleccionar la opción **Simulation→ Run Timing Simulation**.

Cread vosotros una simulación propia para el circuito del multiplexor 4-1 que habéis diseñado que no se parezca a la del ejemplo.

### Tarea 4

Vamos a probar el funcionamiento de los visores (displays) *7-segmentos* de la placa. Vamos a seleccionar un visor *7-segmentos* y en él vamos a mostrar la figura del dígito decimal correspondiente al valor binario que tengamos en un grupo de tres *switch*.



Los visores *7-segmentos* tienen definidos 7 segmentos que pueden iluminarse y apagarse de forma independiente. Ello nos permite formar las figuras que deseemos. Así que si queremos formar una figura que se parezca al número **2** deberemos iluminar los segmentos numerados como 0,1,3,4,6 y deberán permanecer apagados los segmentos 5 y 2. Si miramos el manual veremos que cada segmento se ilumina cuando la señal lógica que lo controla vale 0 y se apaga cuando vale 1.

Cread un proyecto en VHDL que implemente el conversor del número binario que le llega a través de tres bits procedentes de unos *switch* y muestre la figura correspondiente en el visor. Así, si el valor binario es el 000 deberá mostrar el símbolo **0**, si es 001 deberá mostrar el **1** ...

A continuación se muestra el código que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Tarea4 IS
    PORT( SW    : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END Tarea4;

ARCHITECTURE Structure OF Tarea4 IS
BEGIN
    .
    . -- Aquí debéis poner vuestro código que implemente la tarea
    .
END Structure;

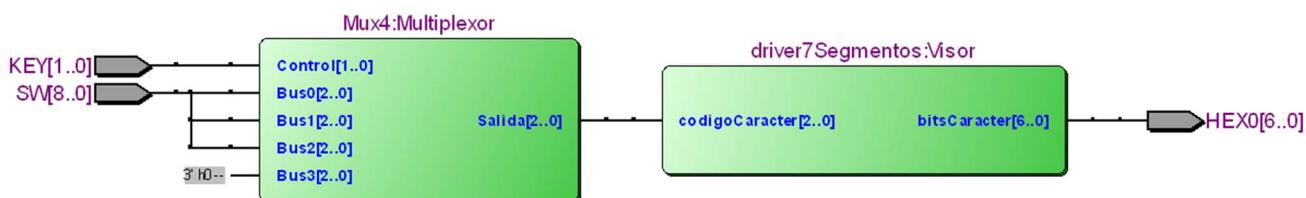
```

### Tarea 5

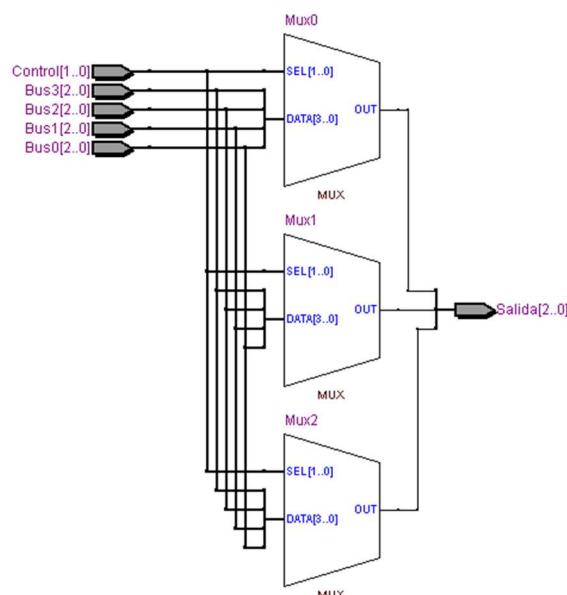
Una vez realizado este conversor vamos a realizar otro que en vez de mostrar números muestre los caracteres de la palabra **HOLA**. Para ello cuando reciba por la entrada el valor 000 deberá mostrar el carácter “H”, cuando reciba el valor 001 deberá mostrar el carácter “O”, cuando reciba el valor 010 deberá mostrar el carácter “L” y cuando reciba el valor 011 deberá mostrar el carácter “A”. Si recibe cualquier otro valor deberá mostrar un guion **-**.

Además, para este visor, el valor que mostraremos podrá proceder de varias fuentes. En nuestro caso procederán de 9 *switch* (agrupados de tres en tres) como señales de entrada. De modo que necesitaremos un multiplexor de 4 entradas de buses de 3 bits (del que sólo usaremos 3 entradas) para seleccionar la fuente con que iluminaremos el visor.

Para implementar toda la tarea usaremos tres módulos (entidades) distintos: Uno será el multiplexor de buses, otro será el conversor del código del carácter a mostrar a los bits que encienden o apagan los segmentos del visor, y por último un módulo que los unirá y los conectarán con los *switches*, los pulsadores y el visor 7 segmentos. El aspecto que tendrá el sistema deberá ser parecido a este.



Cread un nuevo fichero en vhdl que contenga un módulo (entidad) que implemente un multiplexor similar al de la figura.



A continuación se muestra la cabecera del código que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Mux4 IS
    PORT( Control : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          Bus0      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          Bus1      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          Bus2      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          Bus3      : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          Salida   : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END Mux4;

ARCHITECTURE Structure OF Mux4 IS
BEGIN
    .
    .
    . -- Aquí debéis poner vuestro código que implemente el multiplexor de buses
    .
    END Structure;

```

Cread un nuevo fichero en vhdl que contenga un módulo (entidad) que haga la conversión del código del carácter a mostrar a los valores de los 7 segmentos del visor.

A continuación se muestra la cabecera del código que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY driver7Segmentos IS
    PORT( codigoCaracter : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
          bitsCaracter   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END driver7Segmentos;

ARCHITECTURE Structure OF driver7Segmentos IS
BEGIN
    .
    . -- Aquí debéis poner vuestro código que implemente el driver
    .
END Structure;

```

Ahora vamos a crear una entidad que une los dos módulos creados anteriormente. Esta entidad conectará dos pulsadores a las señales de control del multiplexor y conectará a las tres entradas de menor peso del multiplexor los *switches* que codifican el carácter a mostrar (agrupados convenientemente). También conectará la salida del multiplexor a la entrada del conversor y la salida de éste al visor que se encarga de mostrar el carácter.

A continuación se muestra el código completo de esta entidad.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Tarea5 IS
    PORT( SW      : IN STD_LOGIC_VECTOR(8 DOWNTO 0);
          KEY     : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          HEX0    : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END Tarea5;

ARCHITECTURE Structure OF Tarea5 IS
    COMPONENT Mux4 IS
        PORT( Control : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
              Bus0    : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
              Bus1    : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
              Bus2    : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
              Bus3    : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
              Salida  : OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
    END COMPONENT;

    COMPONENT driver7Segmentos IS
        PORT( codigoCaracter : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
              bitsCaracter   : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
    END COMPONENT;

    SIGNAL bus_enlace : STD_LOGIC_VECTOR(2 DOWNTO 0);

BEGIN
    Multiplexor : Mux4
        Port Map( Control => not KEY,
                  Bus0 => SW(2 downto 0),
                  Bus1 => SW(5 downto 3),
                  Bus2 => SW(8 downto 6),
                  Bus3 => "111",
                  Salida => bus_enlace);

    Visor : driver7Segmentos
        Port Map( codigoCaracter => bus_enlace,
                  bitsCaracter => HEX0);
END Structure;

```

Ahora podemos comprobar si todo ha ido bien. Ponemos en los interruptores diversos valores y con los pulsadores vamos seleccionando el valor que debe mostrarse por el visor.

### Tarea 6

En esta tarea vamos a usar los cuatro visores de la placa para mostrar la palabra **HOLA**. Luego usaremos un pulsador para que la palabra rote, carácter a carácter, hacia la izquierda cada vez que pulsemos el pulsador. Para ello usaremos un contador de tres bits que cada vez que pulsemos el botón se incrementará en una unidad módulo 8. El valor de este contador nos indicará que secuencia de caracteres deberán mostrarse por los visores. La siguiente tabla nos muestra qué caracteres mostraremos por los 4 visores en función del valor del contador

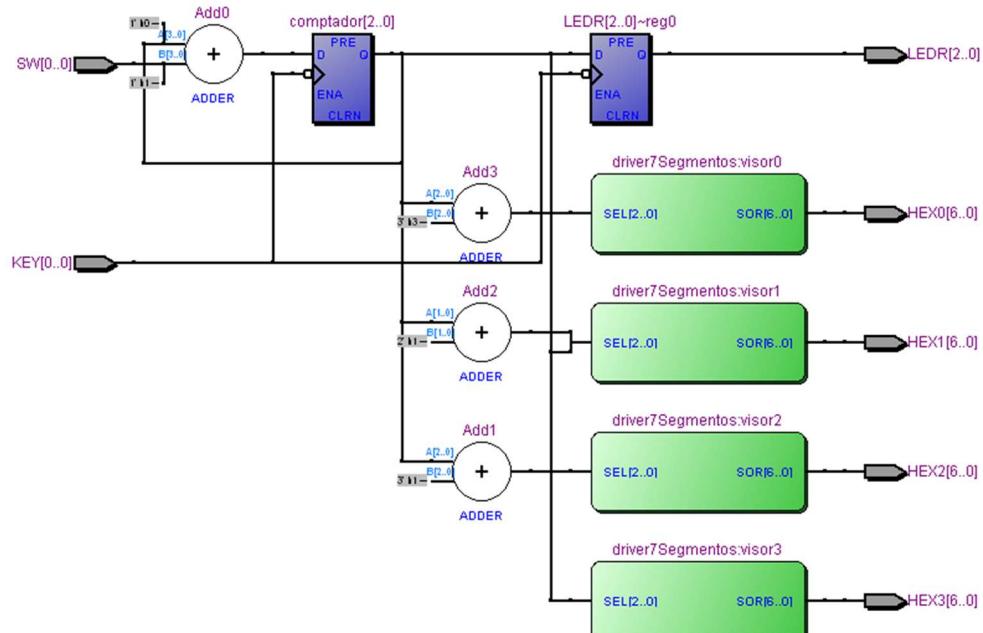
Contador	VISORES
000	H O L A
001	O L A -
010	L A - -
011	A - - -
100	- - - -
101	- - - H
110	- - H O
111	- H O L

Cread un proyecto en VHDL que implemente esta tarea. Usaremos el pulsador cero (*KEY[0]*) como señal para indicar la rotación. Os puede ser útil que los tres leds rojos de menor peso (*LEDR[2..0]*) muestren en todo momento el valor que tiene el contador.

Aunque toda esta tarea se podría hacer fácilmente con una sola entidad debido a su simplicidad, os recomiendo que aprovechéis para practicar múltiples instanciaciones de una misma entidad. Así podéis usar 4 instanciaciones de la entidad *driver7segmentos* que habéis hecho en la tarea anterior. Cada una se encargará de un visor.

Una vez hecho esto, podemos hacer una variación. Vamos a permitir que se pueda rotar la palabra hacia ambos sentidos. Usaremos el interruptor 0 (*SW[0]*) para indicar el sentido de la rotación (0 hacia la izquierda y 1 hacia la derecha) y el pulsador cero (*KEY[0]*) para indicar cuando hay que desplazar los caracteres una posición. Implementad también esta variación.

Si habéis usado múltiples instanciaciones del *driver7Segmentos*, el aspecto que tendrá el sistema deberá ser parecido a este.



A continuación se muestra la cabecera del código de la entidad principal que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Tarea6 IS
    PORT( KEY      : IN std_logic_vector(0 downto 0);
          SW       : IN std_logic_vector(0 downto 0);
          HEX0    : OUT std_logic_vector(6 downto 0);
          HEX1    : OUT std_logic_vector(6 downto 0);
          HEX2    : OUT std_logic_vector(6 downto 0);
          HEX3    : OUT std_logic_vector(6 downto 0);
          LEDR    : OUT std_logic_vector(2 downto 0));
END Tarea6;

ARCHITECTURE Structure OF Tarea6 IS
BEGIN
    .
    . -- Aquí debéis poner vuestro código que implemente la tarea
    .
END Structure;

```

### Tarea 7a

En esta tarea vamos a aprender a usar los relojes de la placa. Primero vamos a diseñar e implementar un circuito que muestre los valores del 0 al 9 por orden en un visor *7-segmentos*. Una vez lleguemos al dígito 9 volvemos a empezar. Cada dígito deberá ser mostrado durante un segundo aproximadamente.

La placa base sólo dispone de una señal de reloj a 50 MHz (*CLOCK\_50*) que es la que usaremos. Por lo tanto, si queremos una señal a 1Hz deberíamos dividir la frecuencia del reloj de entrada por el valor que corresponda (en este caso 50.000.000). Como tampoco deseamos un reloj muy preciso, la forma más fácil de implementar el divisor es haciendo un contador de 25 bits que se incremente en una unidad a cada ciclo de reloj. De modo que el bit de peso 0 del contador es una señal periódica similar a la del reloj que cambia a una frecuencia de 25 Mhz (se ha dividido la frecuencia de entrada por 2), el bit de peso 1 del contador corresponde a una señal periódica de frecuencia 12,5 Mhz, etc. El bit de mayor peso del contador, el 24, es una señal periódica con la frecuencia de 1,5 Hz aproximadamente.

A continuación se muestra la cabecera del código de la entidad principal que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Tarea7a IS
    PORT( CLOCK_50 : IN std_logic;
          HEX0      : OUT std_logic_vector(6 downto 0));
END Tarea7a;

ARCHITECTURE Structure OF Tarea7a IS
BEGIN
    .
    . -- Aquí debéis poner vuestro código que implemente la tarea
    .
END Structure;

```

## Tarea 7b

En esta tarea vamos a implementar la señal de reloj de 1 Hz de forma mucho más precisa. Para ello, simplemente deberíamos inicializar un contador con el valor 50.000.000 e ir restando una unidad a cada ciclo de la señal de reloj de entrada. Cuando este contador valga cero habrá pasado un segundo exactamente. Entonces deberemos volver a inicializar el contador a su valor original para esperar al siguiente segundo. Si deseamos generar una señal periódica de un segundo de frecuencia deberemos generar una salida que vaya alternando su valor entre 0 y 1 cada medio segundo.

La FPGA de la placa de desarrollo dispone de circuitos específicos capaces de generar múltiples señales de reloj de diversas frecuencias (incluso superiores a la nominal), con desfases precisos entre ellas y muchas otras características. Estos circuitos son conocidos como PLL (del inglés Phase-Locked Loop) que de momento no usaremos.

Para implementar esta tarea, primero vamos a implementar una entidad que gestione los 4 visualizadores 7 segmentos de forma agrupada. Esta entidad recibirá un número binario de 16 bits y mostrará por los visores la representación hexadecimal del número binario. Esta entidad nos será útil cuando diseñemos el procesador.

Luego implementaremos la entidad que genera la señal de reloj de frecuencia aproximada a 1 Hz a partir del reloj del sistema a 50Mhz. Para que esta entidad sea más versátil vamos a hacer que el número de bits del contador (y el valor de inicialización) sea parametrizable. Así en el momento de la instancia podremos decidir a qué frecuencia queremos la señal de salida. Recordad que para hacer esto en vhdl hay que usar la sentencia GENERIC.

Finalmente implementaremos la entidad de nivel superior que se encargará de usar adecuadamente estas dos entidades que hemos creado y de llevar la cuenta de los segundos que han pasado desde que se ha puesto en marcha el sistema. Esta entidad deberá mostrar en hexadecimal, el número de segundos transcurridos desde el inicio del sistema. Además cada vez que se presione el pulsador KEY[0] deberá hacer un reset del contador poniendo el número de segundos transcurridos a cero.

Por último, para comprobar que el generador de señales de reloj parametrizable funciona correctamente, probad el diseño haciendo que el contador se incremente cada 0,5 segundos y luego cada 2 segundos.

A continuación se muestra la cabecera del código de la entidad principal que debéis completar:

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Tarea7b IS
    PORT( CLOCK_50 : IN std_logic;
          HEX0  : OUT std_logic_vector(6 downto 0);
          HEX1  : OUT std_logic_vector(6 downto 0);
          HEX2  : OUT std_logic_vector(6 downto 0);
          HEX3  : OUT std_logic_vector(6 downto 0));
END Tarea7b;

ARCHITECTURE Structure OF Tarea7b IS
BEGIN
    .
    -- Aquí debéis poner vuestro código que implemente la tarea
    .
END Structure;
```

## Tarea 8

Ahora que ya sabemos generar señales de reloj a la frecuencia que deseemos repetiremos la tarea 6, pero en vez de rotar la palabra ~~HOLA~~ cada vez que se pulse un botón se hará de forma automática. Se rotará un carácter a cada segundo en el sentido que indique el *switch*.

## Tarea 9

En esta tarea vamos poner juntos en un solo diseño la mayoría de componentes vistos anteriormente y alguno nuevo. Algunos de los primeros métodos de comunicación telegráfica estaban basados en el código Morse. Este código usa patrones cortos y largos de pulsos para representar los caracteres de un mensaje. Cada letra está representada por una secuencia de puntos (patrón corto) y guiones (patrón largo). En la siguiente tabla podéis ver la representación en código Morse de las 8 primeras letras del alfabeto.

Carácter	C. Morse
A	· —
B	— · · ·
C	— · — ·
D	— · ·
E	·
F	· · — ·
G	— — ·
H	·· · ·

Diseñad e implementad un circuito que reciba como entrada una de las 8 primeras letras del alfabeto y muestre su conversión al código Morse en un *led* rojo de la placa. El circuito mostrará el código Morse correspondiente al carácter especificado mediante tres *switch* (*SW[2..0]*) de la placa (000 para el carácter A, 001 para el B, 010 para el C, ...). Además tendrá dos pulsadores (*KEY[1..0]*) como entradas. Mientras no se pulse ninguno de los pulsadores el sistema estará en “receso”. Para indicar que el sistema está en reposo se mantendrá encendido el *led* verde *LEDG[0]*. Cuando pulsemos el pulsador *KEY[1]* el circuito deberá mostrar, una única vez, la conversión del carácter especificado en los *switch* iluminando el *led* rojo *LEDR[0]* (y apagando el *led* verde). Para representar los puntos iluminaremos el *led* rojo durante 0,5 segundos y para representar los guiones iluminaremos el *led* rojo durante 1,5 segundos. Entre dos pulsos consecutivos apagaremos el *led* rojo durante 0,5 segundos. Si mientras se está visualizando un carácter pulsamos el pulsador *KEY[0]* abortaremos la secuencia y volveremos al estado de reposo (fijaos por ejemplo que para representar el carácter C necesitamos 5,5 segundos). Mientras se esté visualizando la secuencia del Morse de un carácter supondremos que el valor del *switch* no va a cambiar y tampoco se va a volver a pulsar el pulsador *KEY[1]*. Una vez se haya apagado el *led* rojo que muestra el último patrón del carácter actual, inmediatamente ya se debe poder volver a pulsar el pulsador *KEY[1]* para mostrar de nuevo una secuencia de patrones.

En todo momento, el carácter que está codificado en los tres *switch* (*SW[2..0]*) de la placa deberá mostrarse por el visor *HEX0* (~~A, B, C, D, E, F, G y H~~).

Para implementar esta tarea primero deberemos decidir cómo codificamos el carácter que vamos a mostrar. Podemos codificarlo como un vector de 0 y 1 donde cada símbolo representa un “punto” o un “guion” respectivamente como en la tabla anterior, o como un vector de 0 y 1 donde cada bit significa el estado del led (encendido/apagado) para cada fracción de 0,5 segundos. En función de la codificación y del diseño incluso puede ser necesario implementar 2 señales de reloj (de 0,5 segundos y de 1,5 segundos).

A parte del módulo generador de las señales de reloj, para resolver la tarea lo podemos hacer de bastantes enfoques diferentes. Aquí tenéis tres ejemplos básicos de enfoques distintos para resolver esta tarea:

- a) Crear un módulo que a partir del valor de los *switchs* genere un vector con la codificación del carácter y la longitud de este vector. Crear otro módulo que reciba estos vectores y las señales de los pulsadores y genere la señal que se mostrará en el *led* rojo de la placa.
- b) Crear un módulo parametrizable, con el vector a mostrar y su longitud como parámetros, que genere la señal del *led* a partir del vector. Entonces crear 8 instancias de ese modulo (una para cada codificación de cada carácter de la tabla) y seleccionar mediante un multiplexor controlado por los *switchs* la que se mostrará en el *led* rojo de la placa.
- c) Crear un grafo de estados, que en función del valor de los *switchs* y la señal de inicio del pulsador, siga un camino del grafo que genere la señal que controla el *led* rojo de la placa que visualiza el morse del carácter seleccionado.

## Tarea 10

Implementando circuitos, a veces se cometen errores que son difíciles de detectar. El diseño se graba perfectamente en la FPGA pero luego no hace lo que deseamos. No es posible debugar paso a paso ni es fácil poner “chivatos” para ver lo que está haciendo el circuito. Para poder analizar que está haciendo el circuito disponemos de una potente herramienta similar a un analizador lógico. Esta herramienta es el *SignalTap II*.

Un analizador lógico normalmente es un dispositivo hardware que nos permite medir las señales que pasan por los buses de un circuito ya construido. Para ello debemos de tener acceso físico a los buses. En el caso de *SignalTap II* se trata de una herramienta software que nos permite hacer lo mismo pero a partir de un diseño grabado en nuestra FPGA. Así pues, resulta muy útil su uso para debugar circuitos y ver cómo funciona exactamente nuestro diseño.

En esta sección veremos con un ejemplo muy simple como utilizar el analizador lógico *SignalTap II*, que viene integrado en el banco de herramientas de *Quartus II*.

En primer lugar abriremos *Quartus II* y crearemos un proyecto para nuestro modelo de FPGA, tal y como hemos hecho otras veces, e incluiremos un nuevo fichero VHDL con el siguiente código. No os olvidéis de cambiar el nombre de la entidad por el de vuestro proyecto.

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY test_analizador IS
  PORT (    CLOCK_50      : IN STD_LOGIC;
            KEY          : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            SW           : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
            LEDR         : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END test_analizador;

ARCHITECTURE Structure OF test_analizador IS
BEGIN
  process (CLOCK_50) begin
    if (rising_edge(CLOCK_50)) then
      LEDR(1) <= SW(1);

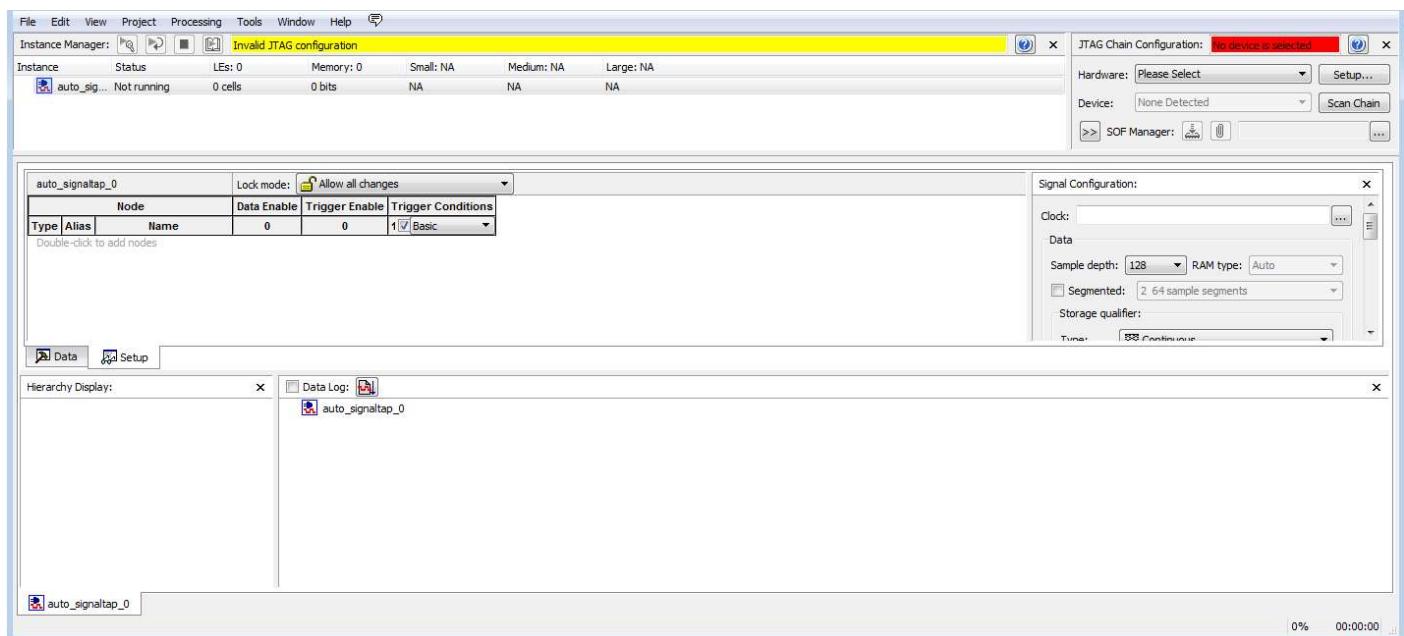
      if (SW(0) = '0') then
        LEDR(0) <= not(KEY(0)) and not(KEY(1));
      else
        LEDR(0) <= not(KEY(0)) or not(KEY(1));
      end if;
    end if;
  end process;
END Structure;

```

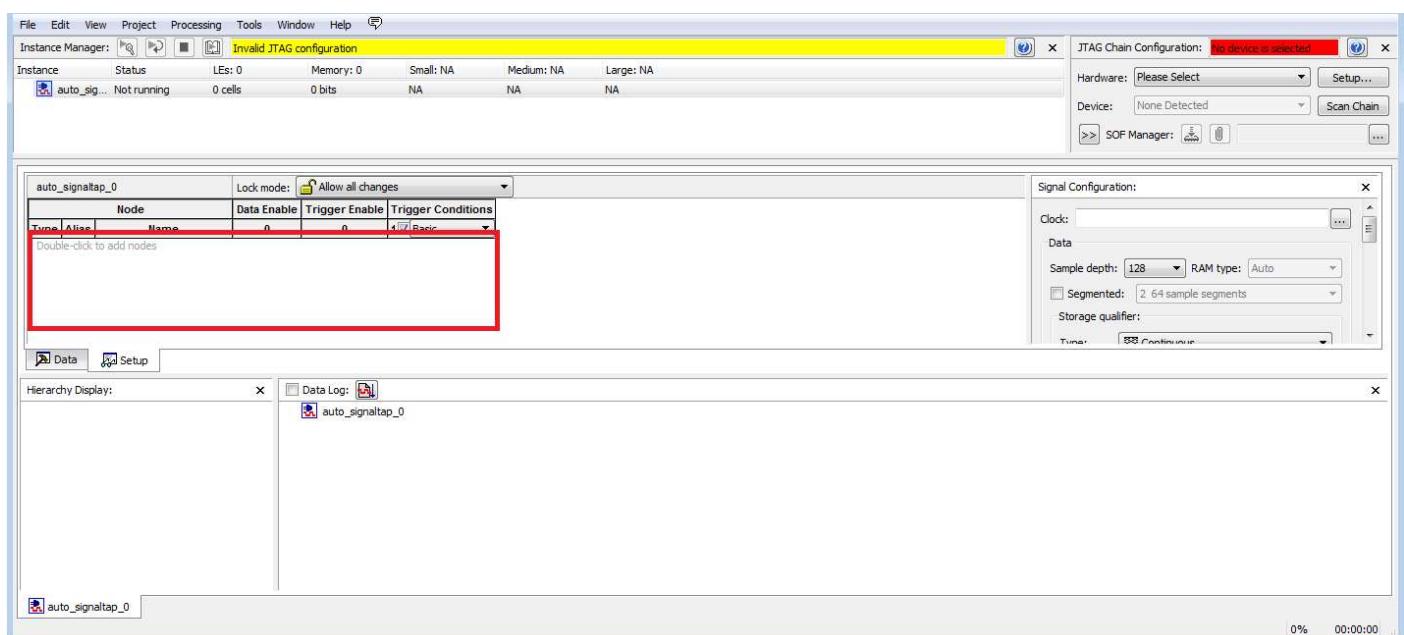
Como podemos ver, este diseño asigna al LED rojo 1 el valor del *switch* 1, y al LED rojo 0 la AND o la OR de los botones 0 y 1, en función de la posición del *switch* 0. Los botones son negados por claridad, ya que un botón pulsado genera un 0. Nótese el uso de la señal de reloj, que aunque no es necesaria para este diseño si lo es para el uso del analizador lógico.

Una vez guardado el código, haremos la asignación de *pins* tal y como hemos hecho siempre, **Assignments** → **Import Assignments...** y buscaremos el fichero de asignaciones a *pins*. Finalmente podemos compilar el proyecto y grabarlo en la FPGA para comprobar que funciona como se espera.

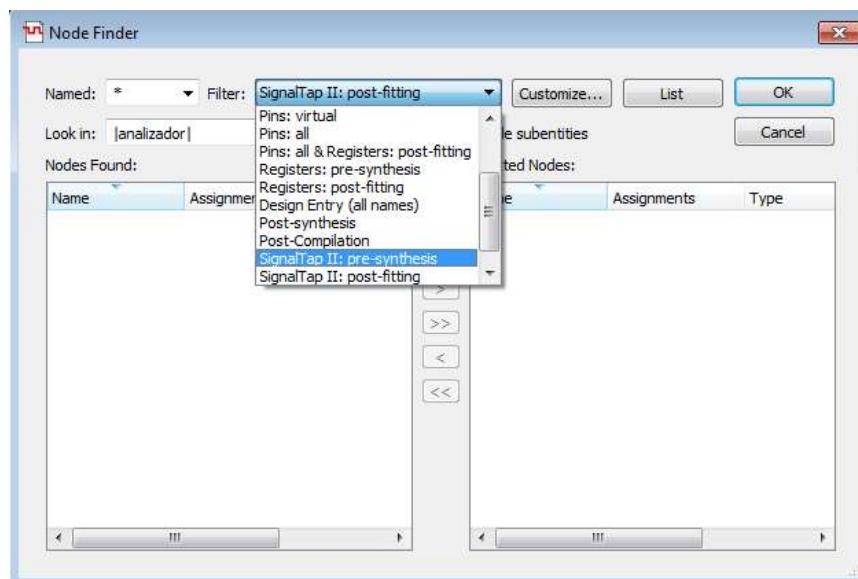
Cuando estemos seguros de que funciona correctamente, en el *Quartus II* escogeremos la opción del menú **File** → **New...**, seleccionaremos *SignalTap II Logic Analyzer File* y pulsaremos **OK**. Entonces se abrirá la siguiente ventana:



Lo primero que debemos hacer es seleccionar las señales que queremos analizar. Para ello haremos un doble-click con el ratón en la zona etiquetada como “*Double-click to add nodes*”.



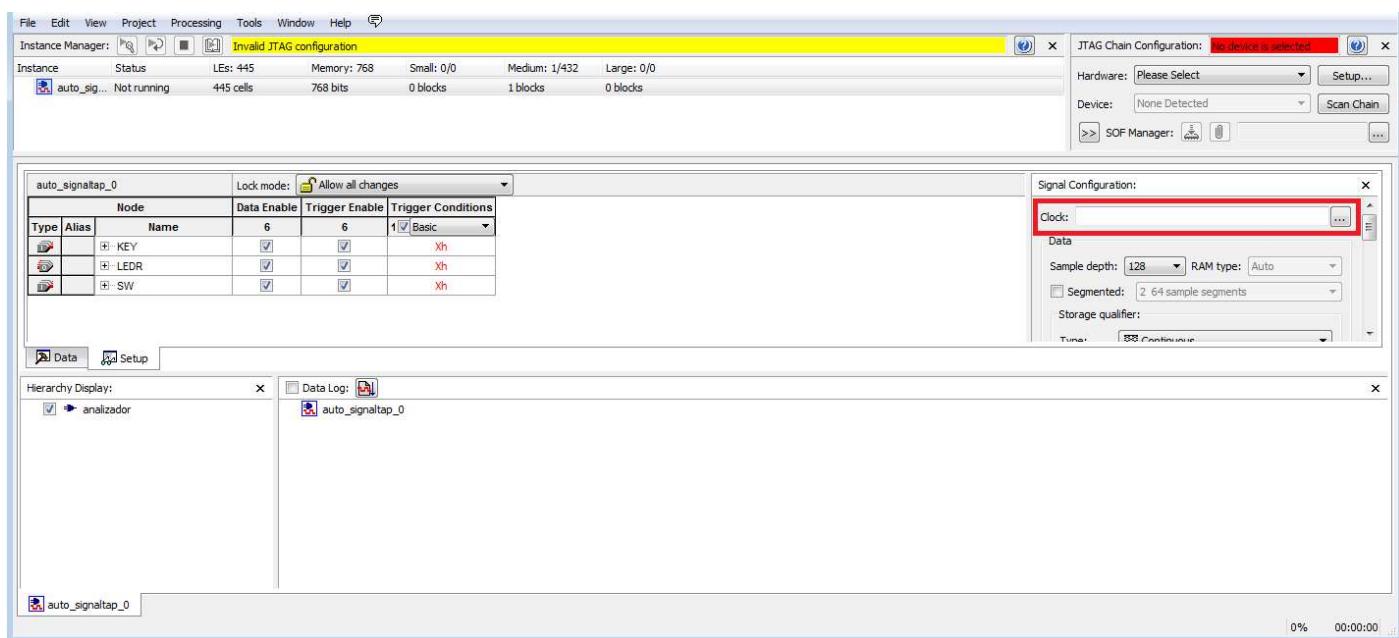
Una ventana de título “*Node Finder*” se abrirá. En esta, desplegaremos el menú etiquetado con “*Filter:*” y seleccionaremos “*SignalTap II: pre-synthesis*”.



A continuación pulsaremos “List” y aparecerá una lista de todas las señales que hemos declarado y que son usadas. Es posible que en ocasiones no aparezcan señales que utilizamos en nuestro diseño, esto es debido a que son optimizadas por el sintetizador porque son redundantes o sus valores no son utilizados. Seleccionaremos las señales *KEY*, *SW* y *LEDR* y pulsaremos el botón “>”. Cuando acabemos, pulsaremos **OK**.

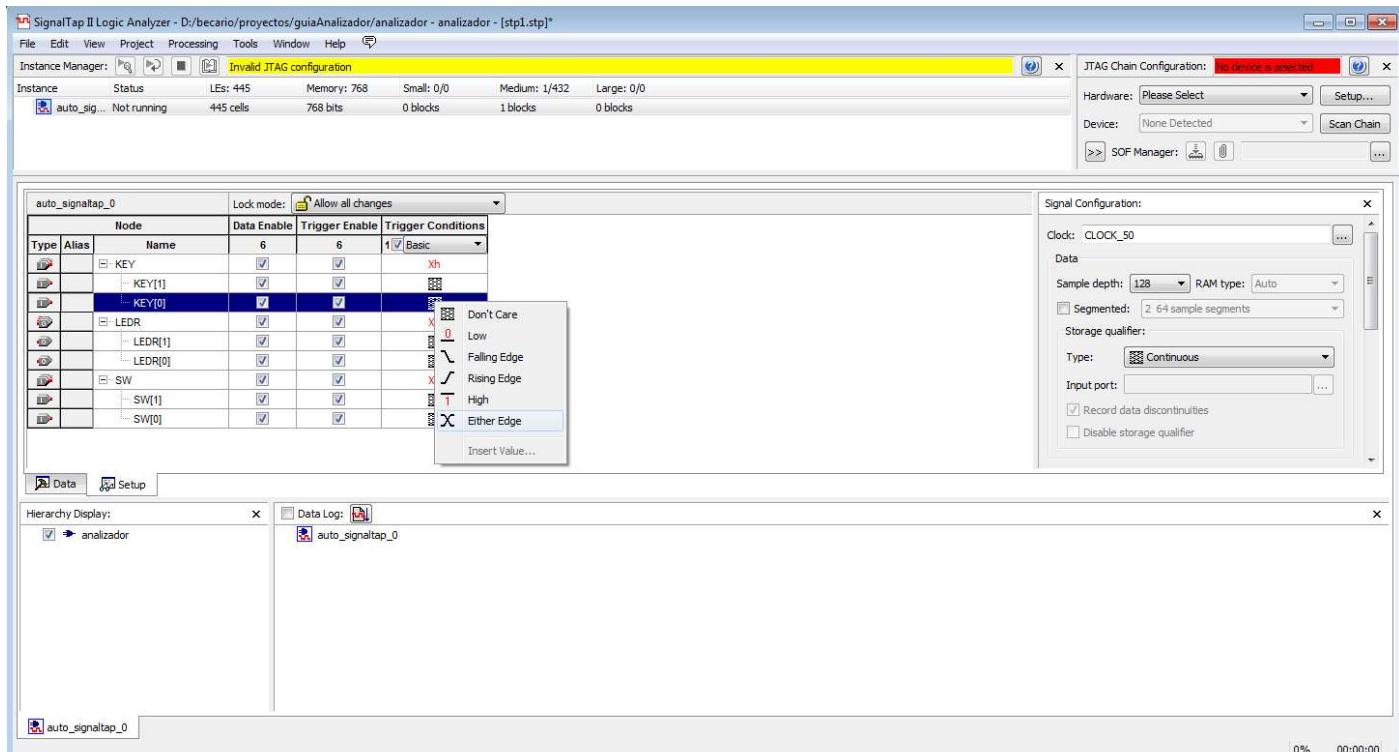


Las señales que hemos añadido habrán aparecido en el recuadro central. Ahora vamos a seleccionar la señal de reloj. En el recuadro de la derecha, etiquetado como “*Signal Configuration*”, pulsa el botón “...” que está a la derecha de la etiqueta “*Clock:*”.



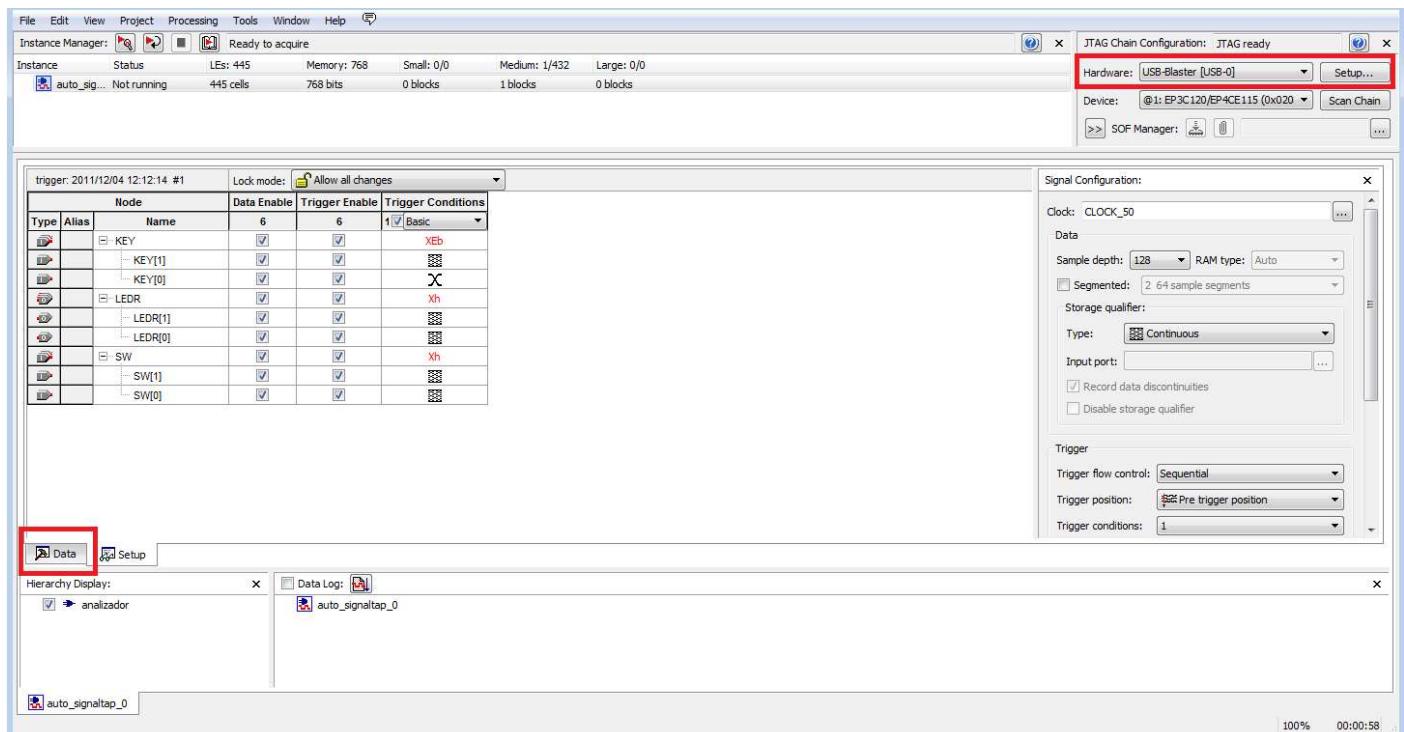
Volverá a aparecer la ventana “*Node Finder*”. Pulsaremos el botón “*List*” y en esta ocasión agregaremos la señal **CLOCK\_50** y pulsaremos **OK**.

Ahora ya podemos empezar a definir las condiciones sobre las que el analizador lógico empezará a almacenar información sobre nuestro circuito. En primer lugar desplegaremos las señales pulsando sobre el botón “+” a la izquierda de cada señal. En la fila de la señal *KEY[0]* y la columna “*Trigger Conditions*” pulsaremos el botón derecho del ratón y seleccionaremos “*Either Edge*”, indicando así que el analizador lógico comenzará a registrar la actividad de las señales cuando *KEY[0]* cambie de valor.

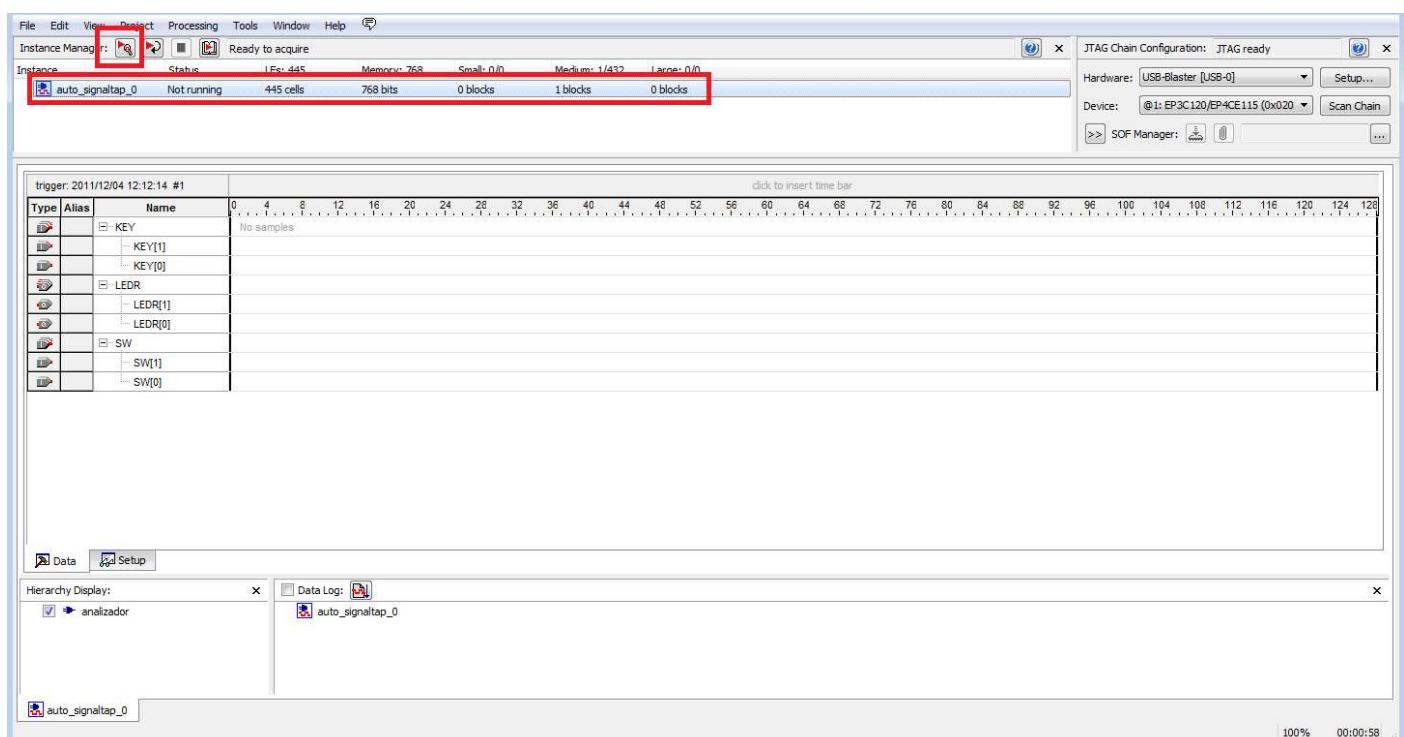


A continuación, cerraremos el **SignalTap II** (**File→Close**) y volveremos a *Quartus II*. Aparecerán varias notificaciones pidiendo guardar los cambios, a las cuales responderemos que sí y guardaremos el archivo generado por *SignalTap II* en la carpeta del proyecto. Finalmente aparecerá un mensaje preguntando si queremos asignar ese archivo al proyecto actual y pulsaremos “**Yes**”. Volveremos a compilar el proyecto y grabaremos el circuito en la FPGA usando el *Programmer*, como hemos hecho siempre.

De vuelta al *SignalTap II* (desde *Quartus II* seleccionaremos la opción de menú **Tools→SignalTap II Logic Analyzer**) seleccionaremos como “*Hardware*” (en la esquina superior derecha de la ventana) el USB-Blaster (es muy probable que ya esté seleccionado). A continuación pulsaremos en la pestaña “*Data*” situada en la esquina inferior izquierda del recuadro en el que aparecen nuestras señales.

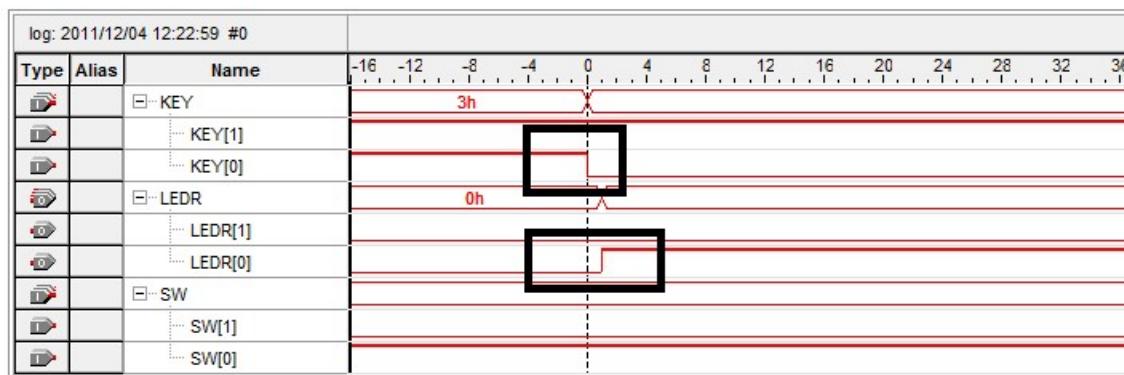


Será en esta ventana en la que veremos la forma que adoptan las señales en el momento en que el valor de *KEY[0]* cambie. Nos aseguraremos de que la instancia *auto\_signaltap\_0* que aparece en el recuadro superior está seleccionada y pulsaremos el icono “Run Analysis”, ambos indicados en la siguiente figura.



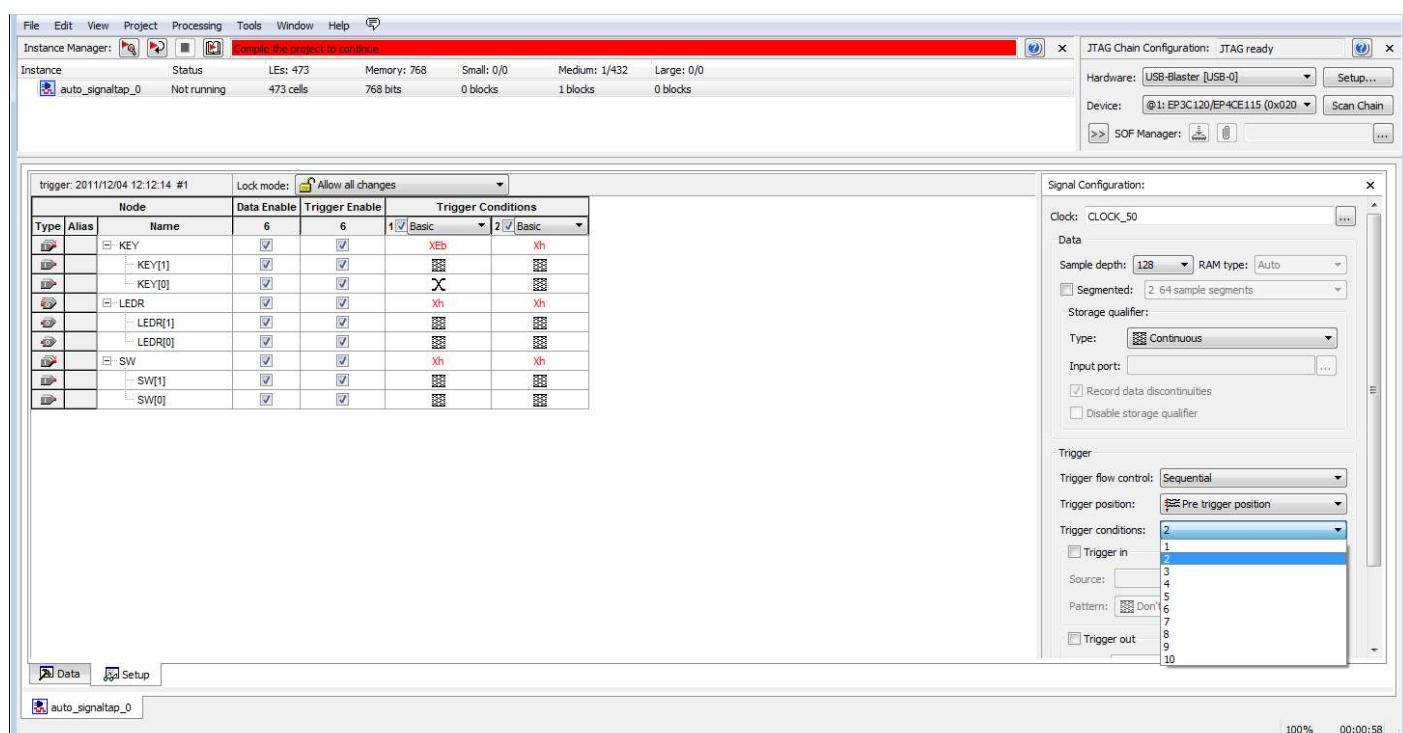
A partir de este momento, un cambio en la señal *KEY[0]* dará lugar a la obtención de la muestra. Podemos desplegar las agrupaciones de las señales *KEY* clicando en el símbolo + para ver en concreto la señal *KEY[0]*. Pulsaremos el botón *KEY[0]* y observaremos el resultado. Veremos que la única señal que cambia de valor es la que representa al propio

botón, ya que al activar esta señal no modifica el valor de ninguna otra. Para obtener una muestra más ilustrativa, moveremos el *switch 0* hacia la posición superior (así se hará una OR de los dos pulsadores en vez de una AND), de manera que al activar el botón 0 se encienda también el led 0. Volveremos a pulsar en el icono de “Run Analysis” y después en el botón 0.



En esta ocasión podremos ver como cambia el valor de la señal *KEY[0]* y un ciclo después el de la señal *LEDR[0]*. Si quisieramos obtener señales continuamente, podríamos pulsar el icono situado justo a la derecha de “Run Analysis”, “Autorun Analysis”, que tomará automáticamente una muestra cada vez que se cumplan las condiciones sin necesidad de pulsar “Run Analysis” a cada muestra. Lo probamos y veremos que cada vez que pulsamos o dejamos de pulsar el botón 0 las señales se actualizan. Para detener la adquisición de datos pulsaremos sobre el icono “Stop Analysis” que está contiguo al de “Autorun Analysis”.

Ahora que ya sabemos tomar muestras, vamos a aumentar la complejidad de las condiciones de activación. Volveremos a la pestaña “Setup” del *SignalTap II*. En el recuadro “Signal Configuration” buscaremos la etiqueta “Trigger conditions” y seleccionaremos el valor 2.



Veremos que aparecerá un recuadro en rojo en la parte superior avisando de que es necesario volver a compilar. Además, habrá aparecido una segunda columna de “Trigger Conditions”. Estas condiciones se deben activar de forma secuencial para que la adquisición de muestras tenga lugar. Por ejemplo, seleccionaremos en la columna de *KEY[0]* como condición 1 “Falling Edge” y como condición 2 “Rising Edge” (si no nos permite hacer cambios es que la adquisición de datos aún

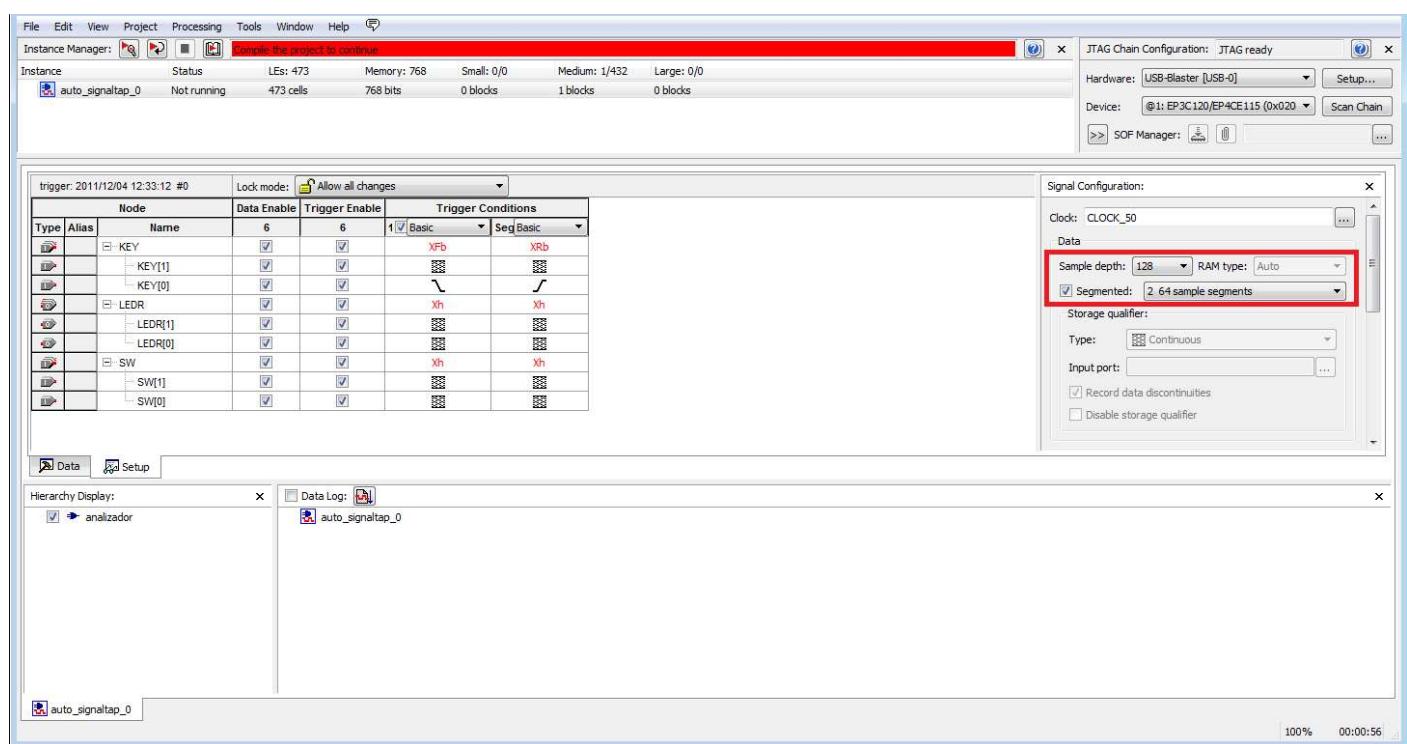
está en marcha). De esta manera, la muestra sólo será tomada cuando la señal *KEY[0]* tenga un flanco descendente seguido de un flanco ascendente, en otras palabras, cuando pulsemos y soltemos el botón 0.

Node			Data Enable	Trigger Enable	Trigger Conditions	
Type	Alias	Name	6	6	1 <input checked="" type="checkbox"/> Basic	2 <input checked="" type="checkbox"/> Basic
RS		KEY	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	XFb	XRb
RS		KEY[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
RS		KEY[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
OD		LEDR	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Xh	Xh
OD		LEDR[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
OD		LEDR[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
RS		SW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Xh	Xh
RS		SW[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
RS		SW[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

Volveremos a compilar y grabar el programa en la FPGA y probaremos que realmente es así como funciona.

Ahora introduciremos la toma de muestras segmentada. Este modo de tomas muestras ocasiona que en lugar de tomar sólo una muestra se tomen varias antes de cargar el valor en el ordenador. A pesar de ello, cada muestra a tomar debe cumplir las condiciones especificadas del mismo modo que si sólo tomáramos una.

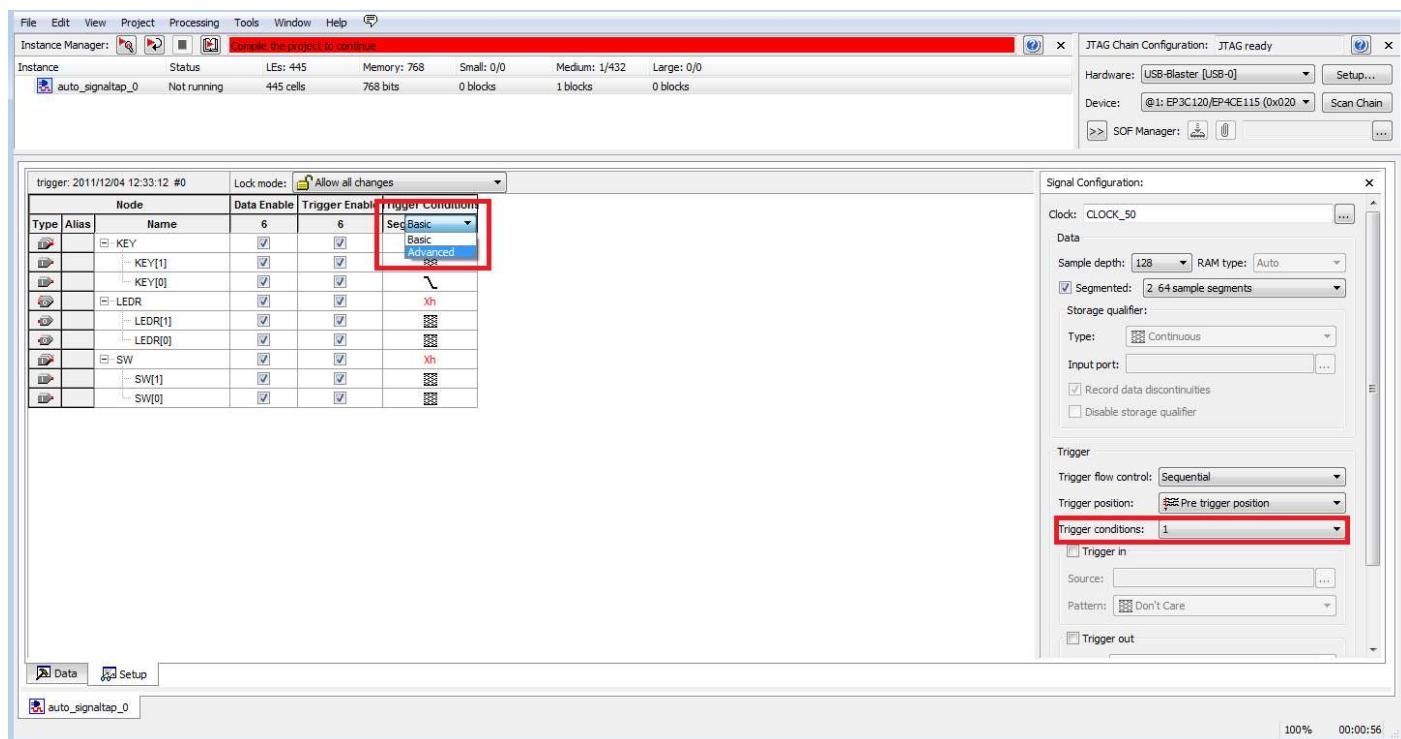
Abriremos el *SignalTap II*. En el recuadro “*Signal Configuration*”, pulsaremos sobre el recuadro marcable “*Segmented*” y seleccionaremos “*2 64 sample segments*” para indicar que queremos obtener 2 muestras de tamaño 64.



Notad también el desplegable “*Sample depth*” que está justo encima. Este nos permite indicar el tamaño de la muestra que queremos obtener.

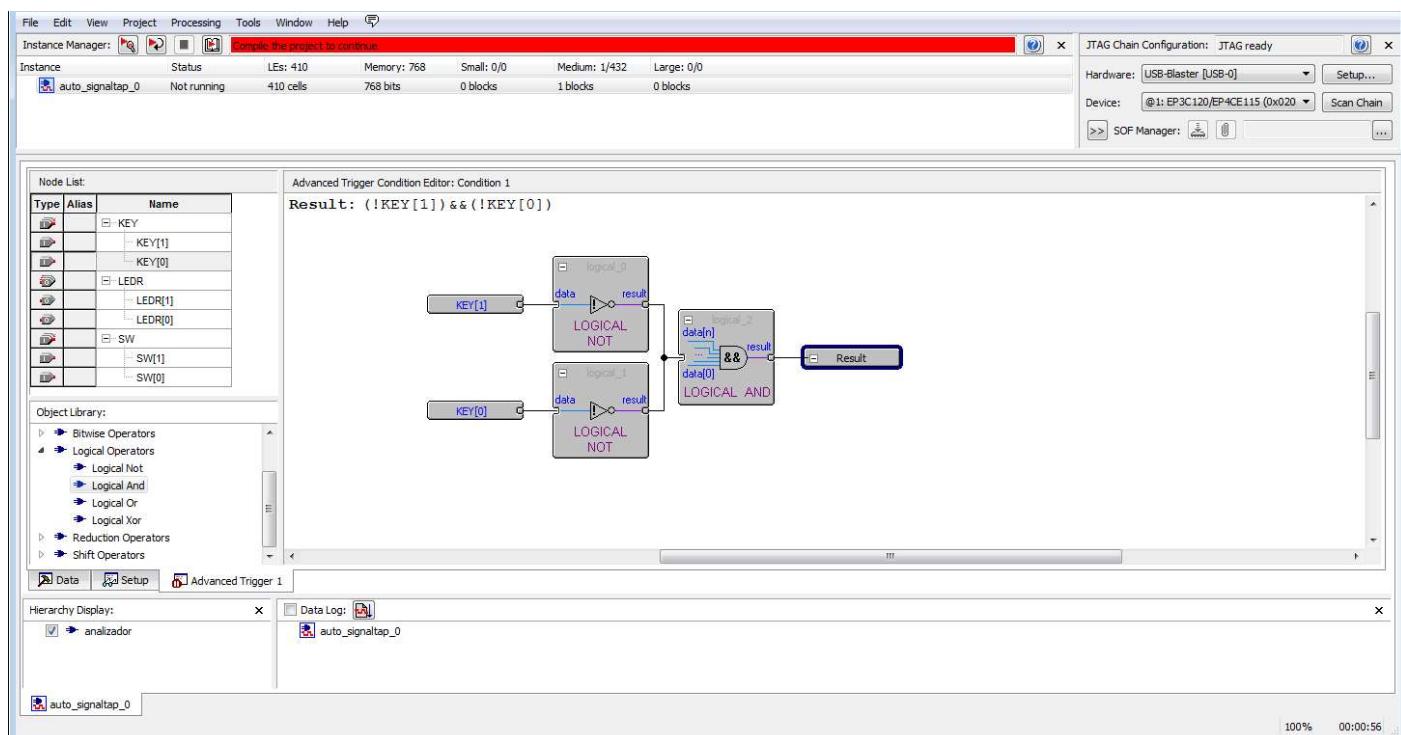
Una vez seleccionada la segmentación, elegiremos las condiciones de *trigger* que queramos. Compilaremos, grabaremos la FPGA y probaremos como necesitamos que estas condiciones se cumplan 2 veces antes de que obtengamos los resultados.

Por último veremos una forma de seleccionar condiciones de activación más complejas. Seleccionaremos que sólo queremos 1 *Trigger Condition*, de manera que sólo aparezca una columna. En la columna de “*Trigger Conditions*” pulsaremos sobre el desplegable “*Basic*” y seleccionaremos “*Advanced*”.



La pantalla cambiará para mostrar el “*Advanced Condition Trigger Editor*”. A la izquierda de la pantalla veremos las señales seleccionadas y justo debajo una librería de objetos. Estos son utilizados para establecer condiciones tan complejas como sea necesario para activar la toma de muestras.

Seleccionaremos primero y luego arrastraremos las señales *KEY[0]* y *KEY[1]* al centro de la pantalla. A continuación buscaremos en la librería de objetos el objeto *Logical Operators* → *Logical Not* y arrastraremos dos al editor. Por último incluiremos también una *Logical And* y las conectaremos como en la siguiente imagen.



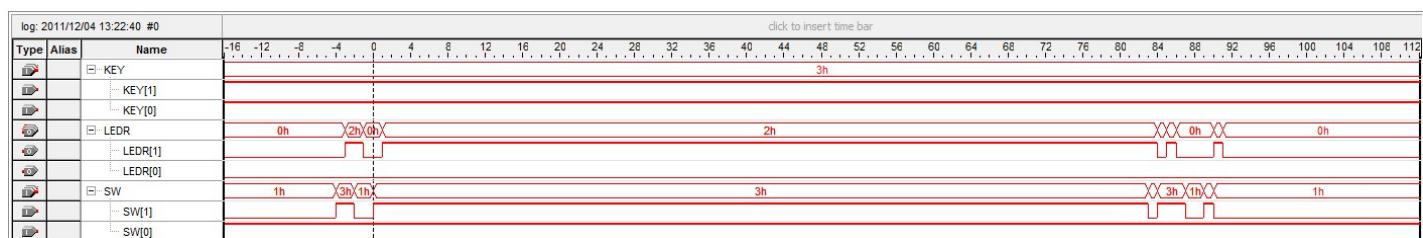
Conectar estos objetos puede ser poco intuitivo al principio. Para conectar dos elementos los acercaremos de manera que aparezca un cable uniéndolos y los soltaremos, no os preocupéis porque queden superpuestos o mal colocados. Cuando conectemos otro elemento y quede superpuesto, el editor se ocupará de separarlos automáticamente pero si intentamos separarlos nosotros lo más probable es que los desconectemos.

Una vez hayamos obtenido el esquema de la figura, volveremos a compilar y a grabaremos el diseño en la FPGA y comprobaremos que la muestra sólo se obtiene cuando mantenemos pulsados los dos botones 0 y 1.

Veremos ahora un ejemplo en el que puede ser muy útil el uso de un Analizador Lógico. Si hubiésemos intentado implementar un contador disparado por uno de los *switch* (el contador debería incrementarse en una unidad cada vez que el *switch* cambiase de 0 a 1) habríamos observado que los resultados obtenidos no son ni mucho menos como cabría esperar. Vamos a ver que pasa utilizando una condición sobre el *switch* 1.

Primero desactivamos la toma de muestras segmentada y dejamos una única condición de *trigger*, que seleccionaremos como “*Basic*”. Al *switch* 1 le asignaremos la condición de *trigger* “*Rising Edge*”. Compilaremos el proyecto, lo grabaremos en la FPGA y miraremos que sucede con esa señal.

A continuación se muestra como ejemplo una posible salida, pero nota que esta no tiene porqué coincidir con la que nosotros obtengamos, ya que hay un importante factor aleatorio.



Estos resultados tan incongruentes se deben a los rebotes mecánicos de los *switches*. Debido a la forma en que están construidos, la señal tarda un tiempo en estabilizarse desde que se desplaza el *switch*. Por ello, al ser leído por un reloj de 50 MHz, podemos ver como la señal baila durante un periodo de tiempo reducido antes de adoptar un valor fijo. Notad que con los botones este fenómeno no sucede, ya que los botones que hay en la placa de desarrollo utilizan filtros para evitarlo.