# Un ejemplo de FPGA
# Síntesis de VHDL

Fernando G. Tinetti*

Reporte Técnico TR-RT-03-2019
III-LIDI, Fac. de Informática, UNLP
CIC, Prov. de Buenos Aires
Argentina
e-mail: fernando@info.unlp.edu.ar
Septiembre 2019

## 1.- Introducción

La idea de este reporte es dejar documentado el proceso de instalación de una placa de desarrollo que contiene un modelo particular de FPGA para sintetizar hardware a partir de la descripción en VHDL (*VHSIC-HDL: Very High Speed Integrated Circuit - Hardware Description Language*). Como parte de este proceso se incluye el IDE (*Integrated Development Environment*) con el cual se trabaja. En resumen, se describirá paso a paso el proceso de:
- Descarga del IDE
- Instalación del IDE
- Instalación de drivers de la placa
- Definición de un proyecto
- Descripción en VDHL del proyecto hasta la síntesis en la FPGA

La placa de desarrollo es conocida como "EP2C5 Cyclone II Mini Board" en [Heller] [wiki], e incluye una FPGA Altera (ahora Intel) Cyclone II. Como en tantos casos similares, la interacción con el hardware específico, que en este caso es la Cyclone II se lleva a cabo en realidad vía esta placa, que tiene varios dispositivos integrados para facilitar no solamente la interacción con la propia FPGA sino para desarrollar aplicaciones/prototios que usan la propia FPGA. La Fig. 1 muestra un diagrama de la placa, que no debe confundirse con una similar, pero para el ambiente educativo como la mencionada en [Intel1] y otras similares, que también incluyen una FPGA Cyclone II. En la Fig. 2 se muestra la placa de desarrollo utilizada junto con el dispositivo de conexión USB.

En particular, el modelo de FPGA Altera Cyclone II utilizado en la placa de la Fig. 2 es el EP2C5T144C8N (impreso en el propio integrado del centro de la placa, que es la FPGA Altera Cyclone II). La fuente de alimentación utilizada es de 5v de cc con Vcc en el centro y con capacidad de proveer 3A (no necesariamente esta placa requiere 3A, se desconoce el requerimiento de corriente, la documentación es muy difícil de encontrar).

En caso de no utilizar una placa de desarrollo, la tarea sería mucho más compleja, directamente trabajando con la propia FPGA, y teniendo en cuenta la documentación de la misma [CycloneII]. En general, para las tareas de desarrollo, experimentación, prototipado, etc. es casi imprescindible contar con una placa de desarrollo como la de la Fig. 2. Por supuesto, todo lo que involucre ser

implementado/sintetizado en la propia FPGA debe tener en cuenta la documentación específica de la misma, es decir la que se provee en [CycloneII].

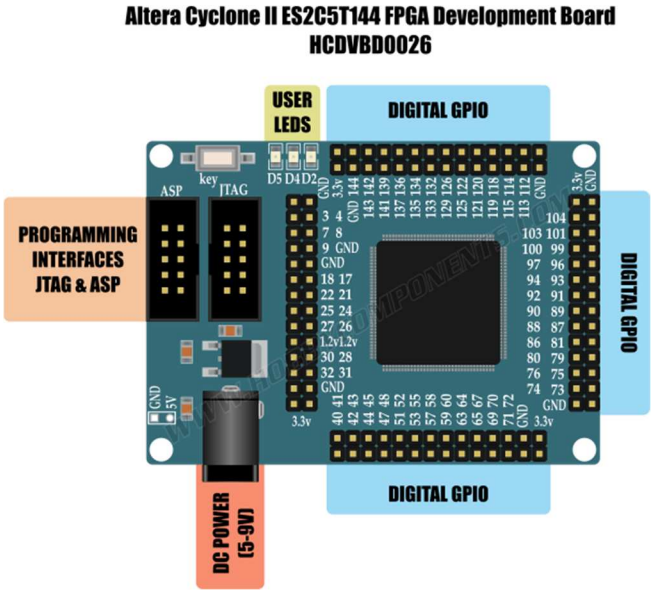


Figura 1: Diagrama de la "*EP2C5 Cyclone II Mini Board*".

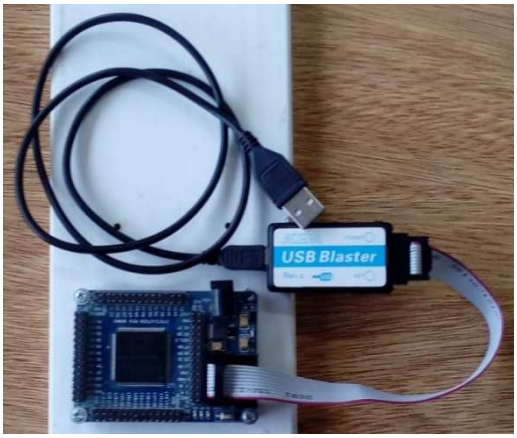


Figura 2: Placa de Desarrollo con el Dispositivo de Conexión "*USB Blaster*".

## 2.- IDE para Síntesis y Driver para Acceder (Windows)

La descripción realizada en esta sección se basa la instalación y uso en Windows, se asume que la instalación y uso en Linux es análoga. El software/IDE recomendado para síntesis en esta placa de desarrollo/FPGA es Quartus Web (en algunas páginas también llamada "Lite") Edition, disponible en el sitio de desacarga de software de Intel para las FPGA [Quartus]. Tiene versiones para Linux y Windows y dos opciones de descarga, una de las cuales es la descarga directa. Se recomienda seleccionar la versión de Quartus de acuerdo a la FPGA a utilizar, que en este caso es la Cyclone

II. Al seleccionar en la página de descarga de acuerdo al dispositivo (según la terminología de a propia página de descarga) se obtiene tanto el propio IDE como lo necesario para sintetizar en la FPGA seleccionada. En este caso, se obtuvieron dos archivos:

- El ejecutable de instalación del IDE: QuartusSetupWeb-13.0.1.232.exe
- La información para síntesis en la Cyclone II: cyclone_web-13.0.1.232.qdz

Tanto la descarga como la instalación (con los valores "default") toman varios minutos, teniendo en cuenta que en total son aproximadamente 2.1GB a descargar y procesar. En particular, la instalación en Windows queda en su valor "default"
C:\altera\13.0sp1

En este punto, y sin tener la placa de desarrollo conectada, se puede ya probar el IDE y llevar a cabo tareas de desarrollo a excepción de la síntesis, por supuesto. En nuestro caso, como el objetivo es documentar hasta la síntesis incluida, directamente se avanza hacia el uso de la propia placa conectada a la PC. De todas maneras, cuando se avance con el ejemplo a sintetizar se verá el uso del propio IDE y se podrá identificar hasta dónde se puede avanzar sin la placa conectada.

Al conectar el USB de la placa de desarrollo, Windows eventualmente (dependiendo de la versión) notificará que no se tiene un driver para manejar el dispositivo conectado, y en cualquier caso dejará constancia en el manejador de dispositivos que no tiene un driver apropiado, tal como se muestra en la Fig. 3.


Fig. 3: Problema de Dispositivo USB Blaster Desconocido.

En este caso, se avanza como con cualquier instalación de driver:
"Update Driver Software…"
==> Browse my computer for…
    ==> Seleccionar c:\altera, que es donde se instaló el IDE (con "include subfolders")
A partir de la búsqueda en ese directorio se muestra que se encontró un driver en particular, y se debe aceptar la instalación de ese driver. Al concluir satisfactoriamente, en el manejador de dispositivos se mostrará que se reconoce la placa de desarrollo, tal como se muestra en la Fig. 4.

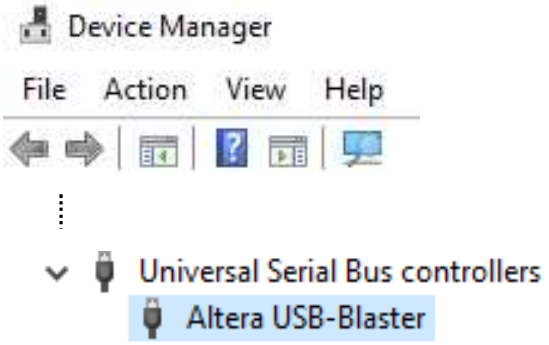


Fig. 3: Driver de USB Blaster Instalado.

## 3.- Primer Proyecto para Sintetizar

Para el primer ejemplo de proyecto a sintetizar utilizaremos un ejemplo muy simple, con el que se podrá recorrer todas las etapas desde el diseño hasta la propia evaluación del diseño, a excepción de la etapa de simulación. Solamente como aclaración: posiblemente la simulación de los diseños sea la tarea en la cual se deba invertir mayor cantidad de tiempo y trabajo, pero queda fuera d este ejemplo y de todo este reporte, que se orienta a ejemplificar la síntesis en una placa de desarrollo en particular.

La Fig. 4 muestra el código VHDL del primer ejemplo, que no es de gran complejidad/ funcionalidad/"*elegancia*", solamente se asegura que los tres leds integrados de la placa utilizada se mantengan apagados. Dado que se piensa directamente en sintetizar el diseño en la placa de desarrollo, necesariamente se deben tener en cuenta detalles de uso de hardware/dispositivos integrados en la placa de desarrollo además de la propia FPGA.

```
ENTITY OffLeds IS
PORT (led0: OUT bit;
      led1: OUT bit;
      led2: OUT bit);
END OffLeds;

ARCHITECTURE OLSynth OF OffLeds IS
BEGIN
  led0 <= '1';
  led1 <= '1';
  led2 <= '1';
END OLSynth;
```

Figura 4. Código VHDL del Proyecto "OffLeds".

En este caso, se controlan los tres leds de la placa de desarrollo y además se tiene en cuenta que cada uno de los leds tiene su cátodo conectado a un pin en particular de la FPGA, con lo cual el led se encenderá o solamente en el caso en el que ese pin tenga el valor correspondiente a GND como salida, es decir '0'. Como se puede esperar, en el código VHDL no se designan los pines específicos, sino que se definen los valores de las señales (en este caso, directamente sobre los ports).

Crear un proyecto en Quartus II es similar a casi cualquier otro IDE (más lo "propio" de este tipo tipo de proyectos, como la definición de la placa FPGA):
File
==> New Project Wizard…
==> Next
==> Seleccionar
a) un directorio para almacenar los archivos del proyecto
b) Nombre del proyecto (el mismo nombre que el de "Entity" para simplificar)
==> Next
==> Next
==> Seleccionar "Family and Device", es decir:
a) Cyclone II
b) EP2C5T144C8 (no aparece ninguno con la "N" final)
==> Next
==> Next

En este punto se muestra un resumen del proyecto creado, tal como se muestra en la Fig. 5, donde quizás para este ejemplo no haya nada muy significativo a excepción del propio nombre del proyecto, "ledpush", y que la familia y nombre del dispositivo para la síntesis son los correctos. La creación se lleva a cabo una vez elegido "Finish" en este reporte de la creación del proyecto.

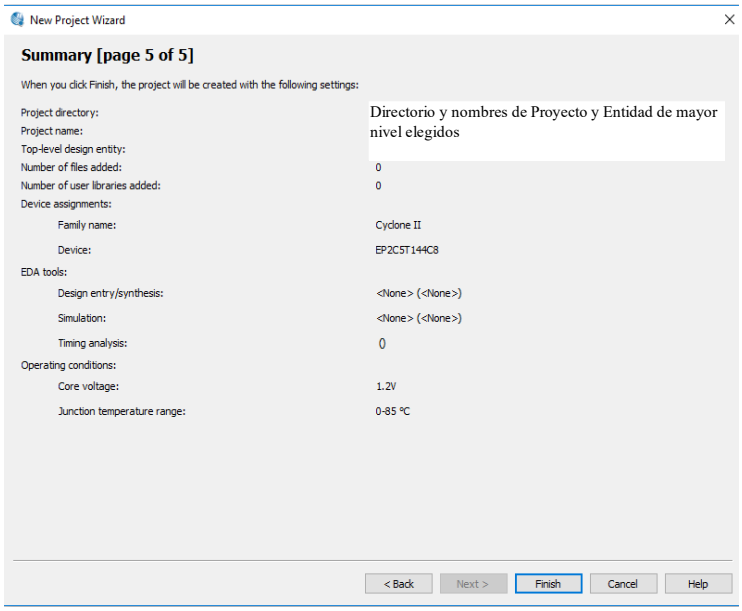


Figura 5: Proyecto "OffLeds" Creado en el IDE.

Este proyecto recién creado no tiene ningún contenido aún, con lo cual debemos agregar en este caso una descripción en VHDL. Para esto, se selecciona
File ==> New
y se presentan varias alternativas, entre las cuales la que utilizaremos es
VHDL File

Se abre entonces un panel o un sector de edición donde se puede incluir el texto de la descripción en VHDL, que será el de la Fig. 4 dada anteriormente. Una vez hecho esto, se puede hacer una primera verificación de la descripción VHDL, al seleccionar "Start Compilation", tal como se muestra en la Fig. 6, de la lista de posibilidades disponibles en el IDE.


Figura 6: Compilación del Proyecto en VHDL.

Se requerirá guardar el archivo de texto y luego hará la compilación, que en este caso es satisfactoria con algunos "*warnings*" (que para este ejemplo se ignorarán, aunque en general se recomienda tenerlos en cuenta). Hasta este punto no se ha hecho nada más que verificar que la descripción en VHDL es correcta.

Avanzando hacia la síntesis, se deben configurar o "asociar" las salidas de la entidad (y las entradas, en el caso en que las haya) "OffLeds", con los pines que corresponda. Para definir esta "asociación" se hace uso de la función "*pin planner*" tal como se muestra en la Fig. 7.

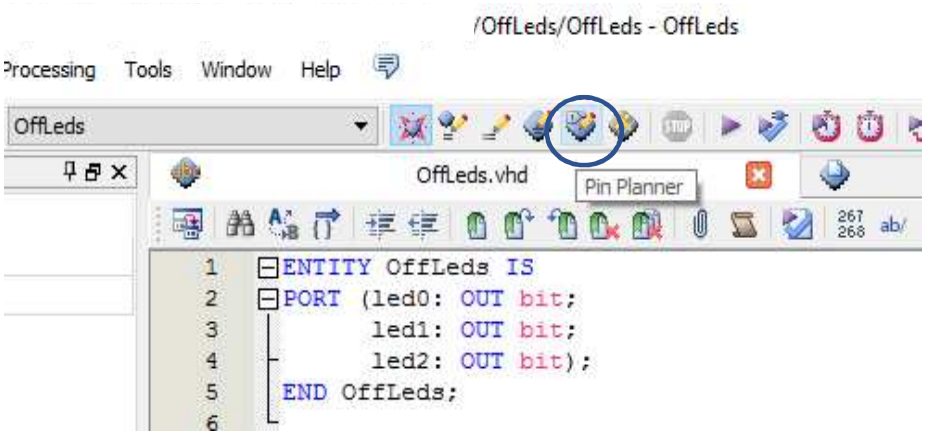


Figura 7. Asignación de Pines, *Pin Planner*.

Al seleccionar esta funcionalidad, se abre una ventana como la de la Fig. 8, donde se pueden especificar los pines, es decir asociar qué pines del dispositivo/FPGA son los que corresponden con la entidad VHDL especificada.

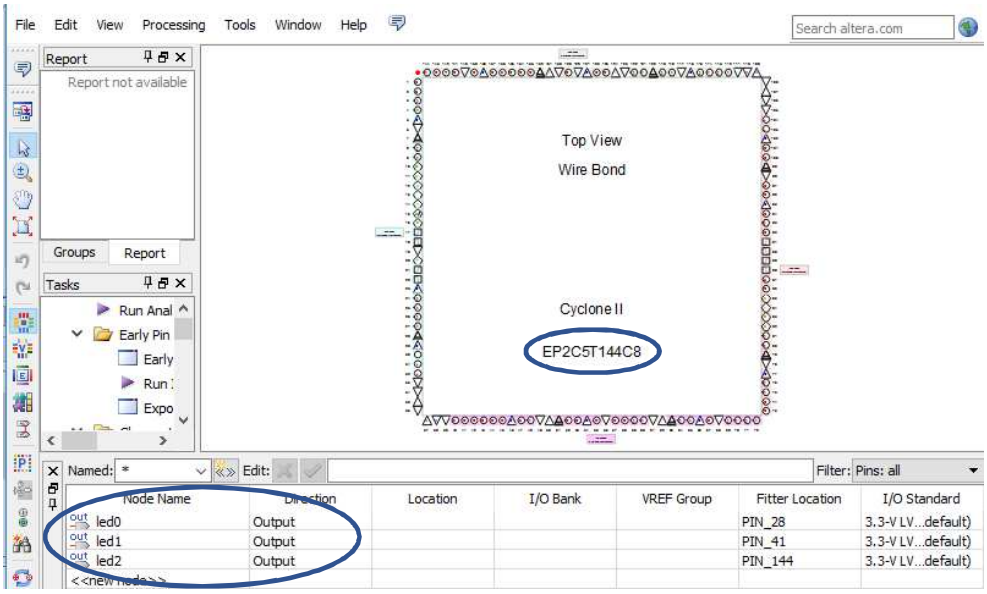


Figura 8: Herramienta de Asignación de Pines.

Entre los detalles más importantes de la asignación de pines se pueden mencionar:

- Los pines a designar son directamente asociados a la FPGA designada como destino de la síntesis del proyecto. En este caso: EP2C5T144C8, que fue indicada de manera explícita cuando el proyecto fue creado.
- La propia herramienta ya identifica las entradas y salidas digitales por los nombres dados en la especificación VHDL del proyecto.
- Se especifican los números de pines, que son los dados en la propia placa, tal como se muestran en la Fig. 1.

En esta etapa se pueden hacer uso de los dispositivos propios de la placa, que se integraron de manera tal que sean accesibles desde la FPGA. Más específicamente, aprovechando lo que se provee en [Heller] y otros sitios que describen esta placa de desarrollo, se pueden utilizar los pines 3, 7 y 9 que están conectados cada uno a un led ya incluido/integrado en la placa de desarrollo. Estas facilidades/dispositivos extra de las placas de desarrollo en general y de esta placa de desarrollo en particular las hacen particularmente útiles para probar, experimentar y prototipar soluciones. En este ejemplo, se designan los pines tal como se muestra en la Fig. 9, es decir:

- led0 conectado/manejando el led del pin 3.
- led1 conectado/manejando el led del pin 7.
- led2 conectado/manejando el led del pin 9.

| Node Name | Direction | Location | I/O Bank | VREF Group |
|---|---|---|---|---|
| led0 | Output | PIN_3 | 1 | B1_N0 |
| led1 | Output | PIN_7 | 1 | B1_N0 |
| led2 | Output | PIN_9 | 1 | B1_N0 |
| <<new node>> | | | | |

Figura 9: Asignación de Pines del Proyecto.

Un detalle físico importante, es que teniendo en cuenta la imagen de la Fig. 1, los pines 3, 7 y 9 se corresponden con los leds identificados en la placa como D2, D4 y D5 respectivamente. Esta información se puede verificar en la propia documentación esquemática de la placa y también directamente probando en la placa de desarrollo misma.

Una vez hecha la asignación, todo lo que hay que hacer es cerrar la ventana de asignación de pines, no hay opción de guardar los cambios, quedan automáticamente asociados/guardados los cambios. Al cerrar la ventana de asignación de pines se recomienda volver a compilar o verificar el diseño utilizando la opción "*compilation*" mostrada en la Fig. 6. Al volver de esta compilación, que no debería generar errores (a lo sumo, *warnings*) se puede (opcionalmente) ver/agregar un panel más en el proyecto, que es el de la asignación de pines, con el título/nombre *Assignment Editor*.

Todas las tareas que se describen en esta sección se pueden llevar a cabo sin la FPGA conectada, es decir sin el USB Blaster conectado a la computadora y/o con la placa de desarrollo sin alimentación. Todo el proceso de definición, compilación/verificación de VHDL y la asignación de pines y su verificación se hace directamente en el IDE sin involucrar a la FPGA. Para el caso utilizado como ejemplo de síntesis en este reporte, no se detallarán simulaciones, que son esenciales en el proceso de desarrollo. Básicamente, sintetizar sin simular es un error, pero no se describirán las tareas de simulación para directamente pasar a sintetizar el diseño hecho en VHDL (que, por otro lado, es muy sencillo). Claramente, la síntesis tiene que ser llevada a cabo sobre la placa de desarrollo misma, afectando la FPGA Cyclone II.

## 4.- Síntesis

Como se explica antes, para sintetizar el diseño en la FPGA, la placa de desarrollo debe estar encendida (en este caso se le proveen 5v de cc, 3A) y conectada a un puerto USB de la computadora con el IDE. El USB Blaster debe estar por un lado conectado a la PC (en un puerto USB), y por el otro lado se puede conectar al puerto JTAG o ASP (*Active Serial Programmer*). El puerto JTAG es el que usaremos en este ejemplo y es el que definitivamente hay que utilizar en el ciclo de desarrollo, dado que permite probar rápidamente los diseños sintetizados en la FGPA. Una vez que el diseño pasa a producción sí hay que sintetizar lo diseñado utilizando el puerto ASP. Lo sintetizado con el puerto JTAG es volátil, automáticamente se pierde al dejar de alimentar la FPGA, lo sintetizado con el puerto ASP no es volátil, no se pierde aunque la FPGA pierda la alimentación, una vez encendida mantiene la funcionalidad de lo sintetizado. En el caso de la placa de desarrollo que se utiliza en este ejemplo, el puerto JTAG es el más cercano a los pines de GPIO y el ASP es el que está en el límite de la placa, más alejado de los pines GPIO, tal como se puede verificar en la Fig. 1.

Para el proceso de síntesis se usa la funcionalidad "*programmer*" del IDE, tal como lo muestra la Fig. 10. Si la placa no está conectada o no se le ha provisto alimentación o incluso la primera vez que se intenta sintetizar el diseño, se tiene indicado que la placa no está conectada o lista para ser utilizada para síntesis, como se muestra en la Fig. 11. En el caso en que esté conectada y encendida, se debe utilizar "Hardware Setup", y en la ventana que se abrirá, como la de la Fig. 12, seleccionar USB-Blaster y Add Hardware.

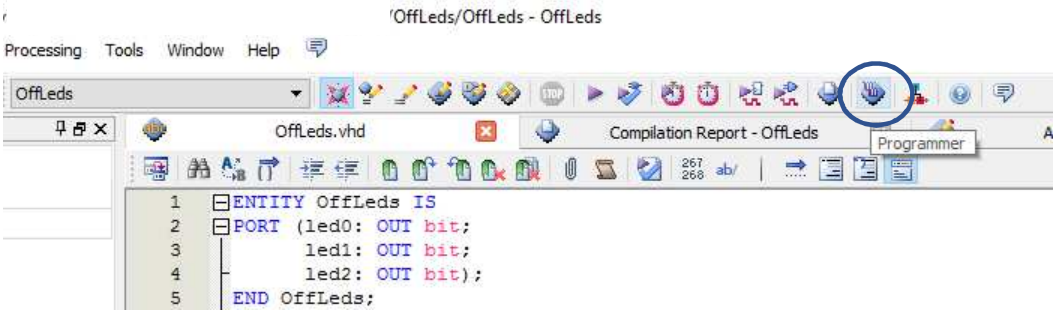


Figura 10: Síntesis, uso de “Programmer”.

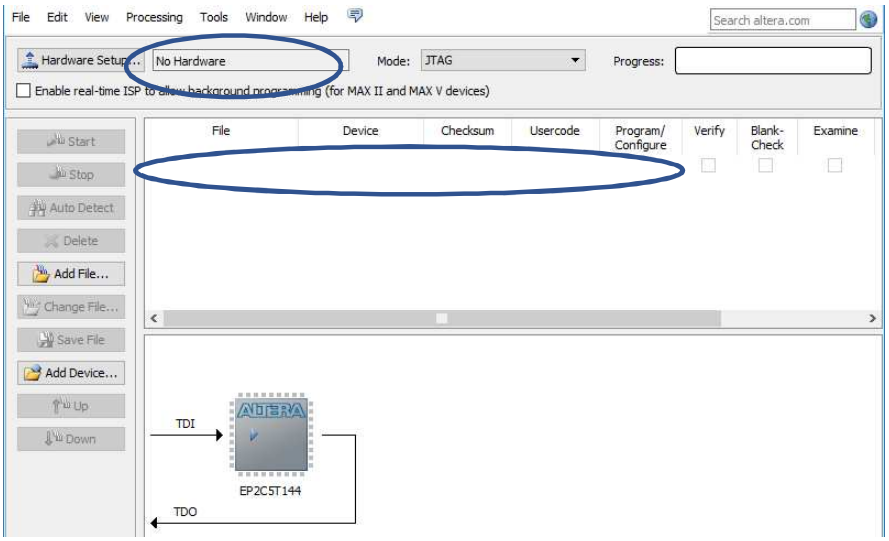


Figura 11: Programador Sin Hardware a Usar.

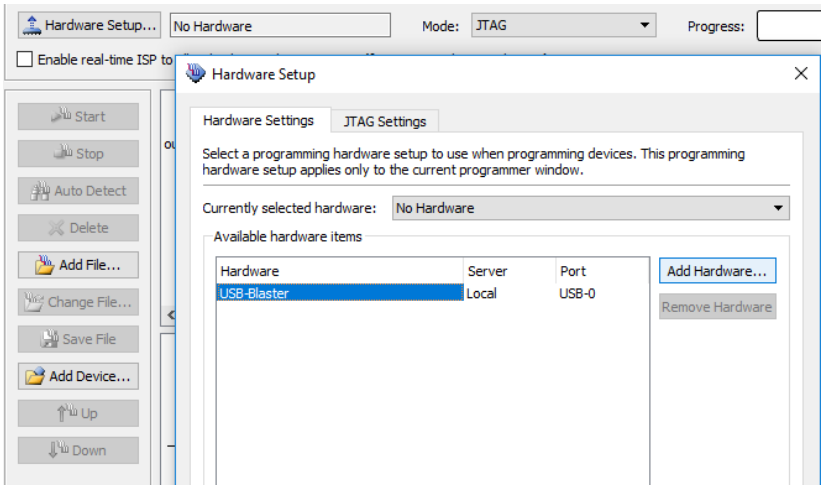


Figura 12: Agregar el Hardware de la Placa para la Síntesis.

También suele ser necesario el agregado del archivo a usar para la síntesis, que debe hacerse con "Add File..." y seleccionando del directorio "output_files" el archivo correspondiente al proyecto y con extensión .sof (el de extensión .pof es necesario cuando se sintetiza vía el puerto ASP). La Fig. 13 detalla toda la información que debe estar definida para sintetizar el proyecto. La síntesis propiamente dicha se lleva a cabo con "*Start*".

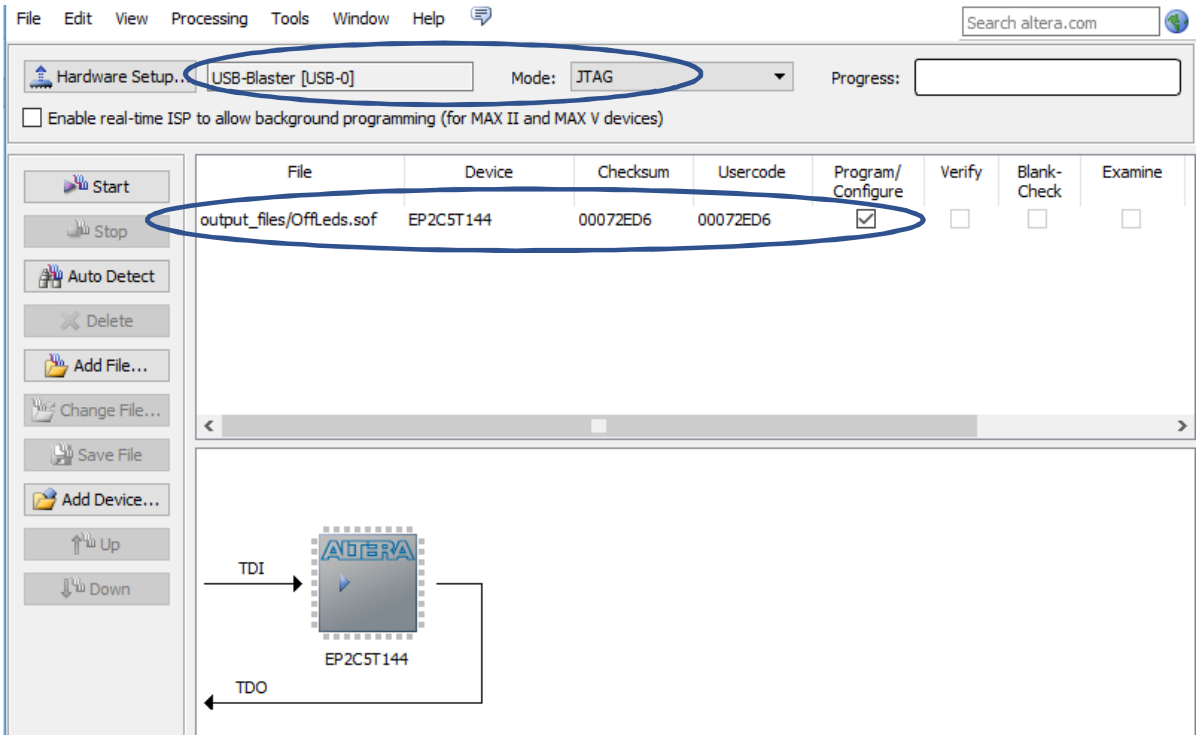


Figura 13: Proyecto Listo para ser Sintetizado.

Cuando en "*Progress*" se tenga "100% (Successful)" la síntesis se habrá completado y el diseño estará en la propia FPGA (en este caso, apagando los 3 leds integrados en la placa y conectados a la FPGA).

Con este diseño tan sencillo se pueden verificar los pines conectados a los leds, de manera tal que es pueden apagar (con '1') y encender (con '0') desde el propio código VHDL. En vez de eso, se continuará con algo más elaborado, aprovechando que la placa cuenta con un botón pulsador que se conecta a uno de los pines de la FPGA. Al cerrar el programador se recomienda guardar lo realizado y luego directamente cerrar el proyecto para comenzar uno nuevo.

## 5.- Entrada de Botón Pulsador y Salidas a Leds

En este proyecto se aprovechará el botón pulsador para controlar el encendido/apagado de uno de los leds y se dejarán los dos leds restantes apagados. Específicamente, el botón pulsador a utilizar es el que ya tiene integrado la propia placa de desarrollo, que al ser presionado conecta con GND. La Fig. 14 muestra el código VHDL para controlar el encendido de un led (recordar que se

encienden cuando se conectan a GND desde la FPGA), donde además se deja como comentario cuál led se controlará con la señal del botón pulsador integrado en la propia placa. Resumiendo:

- Como en el ejemplo anterior, se manejan los tres leds de la placa de desarrollo.
- Los leds de los "extremos" (de la fila de 3 leds), D2 y D5, se dejan continuamente apagados.
- El led del centro, D4, se encenderá cuando se presione el botón pulsador.

```
ENTITY Pushled IS
PORT (pushb: IN  BIT;  -- PIN_144 <==  Push Button
      led0 : OUT BIT;  -- PIN_3    ==> D2 led
      led1 : OUT BIT;  -- PIN_7    ==> D4 led
      led2 : OUT BIT); -- PIN_9    ==> D5 led
END Pushled;

ARCHITECTURE PushledArq OF Pushled IS
BEGIN
  led0 <= '1';          -- OFF
  led1 <= pushb;        -- "On" when button pressed
  led2 <= '1';          -- OFF
END PushledArq;
```

Fig. 14: Encendiendo un led con un Botón Pulsador.

Se debe recordar que el código VHDL no puede hacer referencia explícita a lo que está fuera de la propia FPGA, y de hecho tampoco hace referencia a la FPGA misma. El proceso de síntesis es el que "resolverá" la forma en que el diseño que se describe en VHDL terminará siendo implementado en la FPGA. Por otro lado, la asignación de pines junto con lo específicamente integrado en la placa de desarrollo donde se sintetizará el diseño será lo que determinará de dónde llegará la señal de entrada y hacia dónde irán las señales de salida de la FPGA.

Siguiendo los mismos pasos que para el proyecto anterior, con el agregado de asignar la entrada "pushb" al pin 144 de la FPGA que está conectado al *push button* de la placa de desarrollo tal como se muestra en la Fig. 15, se puede llegar a sintetizar el diseño. La síntesis propiamente dicha se lleva a cabo como en el caso anterior, utilizando la función "*programmer*" del IDE.

| Node Name | Direction | Location |
|---|---|---|
| out led0 | Output | PIN_3 |
| out led1 | Output | PIN_7 |
| out led2 | Output | PIN_9 |
| in pushb | Input | PIN_144 |
| <<new node>> | | |

Figura 15: Asignación de Pines del Proyecto PushLed.

En [video1] se puede ver el funcionamiento de este diseño ya sobre la placa de desarrollo, donde se puede verificar que, efectivamente, los leds D2 y D5 están siempre apagados y D4 se enciende cuando se presiona el botón pulsador de la propia placa. Un detalle importante y podría decirse que casi exclusivamente relacionado con el hardware y las señales involucradas es el que se puede visualizar en el video [video2]:

- El led D4 tiene un encendido cuando solamente se toca el botón pulsador, sin presionarlo.
- El led D4 tiene "otro" encendido cuando se presiona el botón pulsador.

Esto se debe a que, aunque se describa en VHDL en términos de "BIT", lo que se está manejando son las propias señales de la placa y en la FPGA. Si recurrimos al detalle del esquemático de la

placa de desarrollo podremos ver que el botón pulsador es del tipo que se muestra en la Fig. 16, es decir que al ser presionado conectará a GND, pero sin ser presionado queda el circuito abierto, es decir que es muy difícil asegurar ningún valor binario, en realidad.

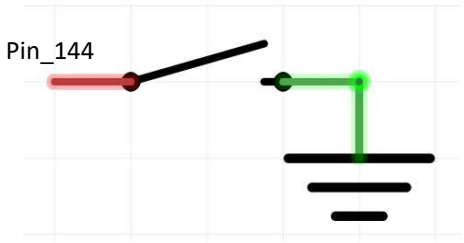


Figura 16: Botón Pulsador con Conexión a GND.

Dado el esquema de la Fig. 16, incluyendo la asignación de pines, básicamente en el pin 144 de la síntesis en FPG se tendrá "lo que se reciba" del botón pulsador, que en el caso en que esté presionado seguro será GND, pero en el caso en que no esté presionado, no necesariamente la señal será Vcc ni algo similar. El problema en este punto es directamente físico, muy difícilmente relacionable con la lógica binaria que se podría asumir del diseño en VHDL, donde
'0' ==> led encendido.
'1' ==> led apagado.

Una posibilidad que se podría interpretar como "más independiente" del circuito abierto sería la especificada como se muestra en la Fig. 17. Sin embargo, al sintetizar esa arquitectura en realidad se mantiene el comportamiento visualizado en [video2].

```
ENTITY Pushled IS
PORT (pushb: IN  BIT;  -- PIN_144 <==  Push Button
      led0 : OUT BIT;  -- PIN_3    ==> D2 led
      led1 : OUT BIT;  -- PIN_7    ==> D4 led
      led2 : OUT BIT); -- PIN_9    ==> D5 led
END Pushled;

ARCHITECTURE ProcArq OF Pushled IS
BEGIN
  PROCESS(pushb)
  BEGIN
    if (pushb = '0') then
      led1 <= '0';
    else
      led1 <= '1';
    end if;
    led0 <= '1';
    led2 <= '1';
  END PROCESS;
END ProcArq;
```

Fig. 17: Proceso para Encendido de un led con un Botón Pulsador.

También como detalle de síntesis de los diseños de la Fig. 14 y la Fig. 17, es interesante notar que el reporte del IDE para la síntesis de estos diseños es idéntico en términos de la cantidad de recursos, que se muestra en la Fig. 18. De hecho, en ambos casos se tienen exactamente la misma cantidad de "avisos"/*warnings*. Aunque la sencillez de la especificación de la Fig. 14 podría hacer suponer que no se usa ningún LE, ya es diferente el caso del diseño de la Fig. 17, con código

denominado “secuencial” en VHDL. En principio, podría esperarse que, por ejemplo, el diseño de la Fig. 17 utilice algún recurso LE (Logic Element) más que el diseño de la Fig. 14, pero en ambos casos solamente se reporta que se utilizan 4 pines. Y estos 4 pines son directamente los directamente especificados para la entidad y, por supuesto, asignados en los correspondientes pines de la FPGA tal como se explica antes. Por lo tanto, la funcionalidad en ejecución real (sintetizada) y también el reporte de utilización de recursos son idénticos para ambos diseños. Sin entrar en detalles y sin tener referencia del “gate level netlist” generado por el IDE se podría suponer, en principio, que se sintetiza exactamente igual, con los mismos recursos/conexiones de/en la FPGA de Altera.

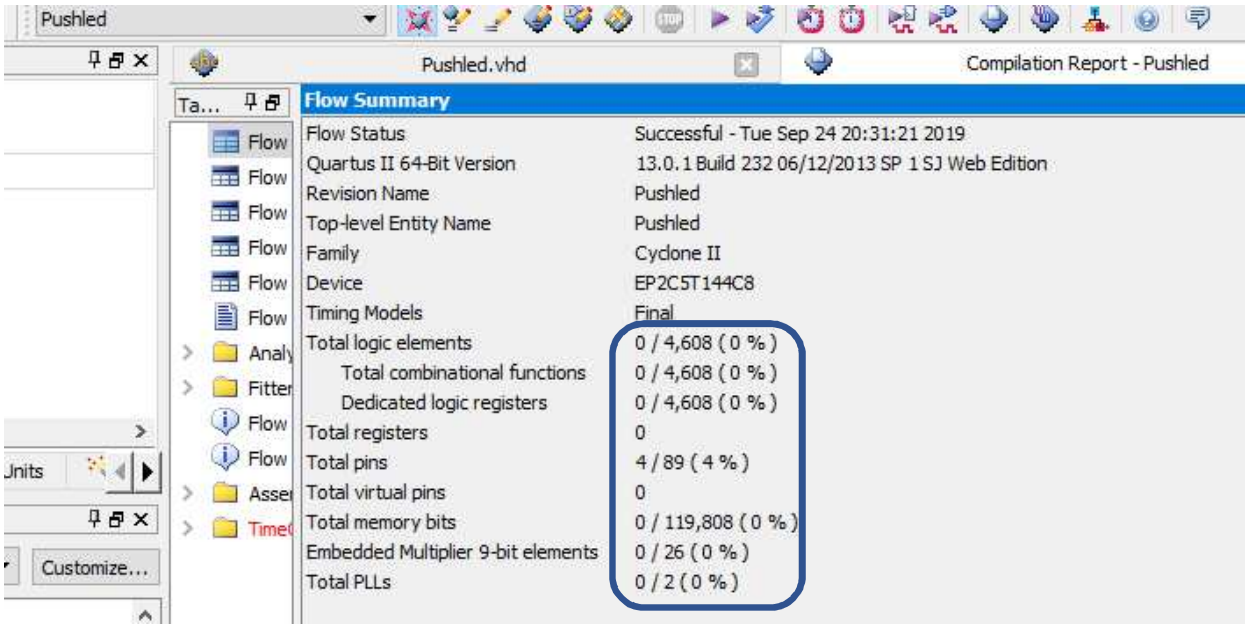


Figura 18: Recursos Utilizados para la Síntesis del Diseño.

En este documento no se avanzará en el detalle de lo que puede estar sucediendo en realidad, solamente se sugiere utilizar otra alternativa de dispositivo botón pulsador o conectar el pin_144 a un I/O pullup para evitar el problema de un circuito abierto (y la respectiva conexión “flotante”, como se suele mencionar).

Finalmente, en [video3] se muestra el diseño de la Fig. 14 funcionando correctamente (el led se enciende solamente en el caso en que se presiona el botón pulsador) gracias a la conexión de un (*weak*) *pullup* al pin 144 de la FPGA. Esto en la práctica genera que el pin_144 efectivamente se lee/interpreta como ‘1’ en el caso en que el botón pulsador no se presiona y se lee/interpreta correctamente como un ‘0’ cuando el botón pulsador se presiona, dado que se tiene conexión a GND en el pin_144.

# 6.- Proyecto con Reloj

La Fig. 18 muestra básicamente el mismo diseño de la Fig. 17, con el agregado de un proceso que cambia la señal “blkval” a su valor inverso cada $50x10^6$ cambios de valor (*rising edge*) de la señal

“clk” que. Esta señal “clk” será asociada, como es de esperarse, a un reloj de 50MHz y la señal blkval será la que determine el valor de uno de los leds (que, a su vez, será encendido o apagado dependiendo del valor). Es posible que existan mejores especificaciones de diseño para obtener el mismo resultado en la síntesis, este caso se proporciona solamente como ejemplo sencillo de aprovechamiento de los recursos disponibles de la placa de desarrollo.

```
entity PL_CLK is
Port (clk  : IN  BIT;  -- PIN_17  <== 50MHz Clock
      pushb: IN  BIT;  -- PIN_144 <==  Push Button
      led0 : OUT BIT;  -- PIN_3    ==> D2 led
      led1 : OUT BIT;  -- PIN_7    ==> D4 led
      led2 : OUT BIT); -- PIN_9    ==> D5 led
END PL_CLK;

architecture PLCLKArq of PL_CLK is
  signal clkcntr  : integer range 0 to 50000000 := 0;
  signal blkvalue : BIT := '0';
  signal led1val  : BIT := '1';

begin
  counter : process(clk)
  begin
    if clk'event and clk = '1' then
      if clkcntr = 49999999 then
        clkcntr  <= 0;
        blkvalue <= not blkvalue;
      else
        clkcntr <= clkcntr + 1;
      end if;
    end if;
  end process;

  pushled : process(pushb)
  begin
    if (pushb = '0') then
      led1val <= '0';
    else
      led1val <= '1';
    end if;
  end process;

  led0 <= blkvalue;
  led1 <= led1val;
  led2 <= '1';
end PLCLKArq;
```

Fig. 18: led Intermitente y led con un Botón Pulsador.

La Fig. 19 muestra la asignación de pines, donde solamente se agrega la entrada del clock de 50MHz integrado en la placa de desarrollo y que está conectado al pin_17 de la FPGA. En [video4] se muestra el diseño sintetizado en la FPGA. En este caso también se tiene conectado un pullup al pin de entrada que a su vez está conectado al botón pulsador (pin_144). Como era de esperar, uno de los leds, específicamente el conectado a led0 del diseño se enciende y apaga intermitentemente y el otro led, el conectado a led1 del diseño se enciende solamente cuando se presionad el botón pulsador de la placa. Además, el led que se enciende con el botón pulsador solamente cambia de estado cuando corresponde, sin tener el problema del circuito abierto al utilizarse el *pullup* para el pin de entrada de la FPGA.

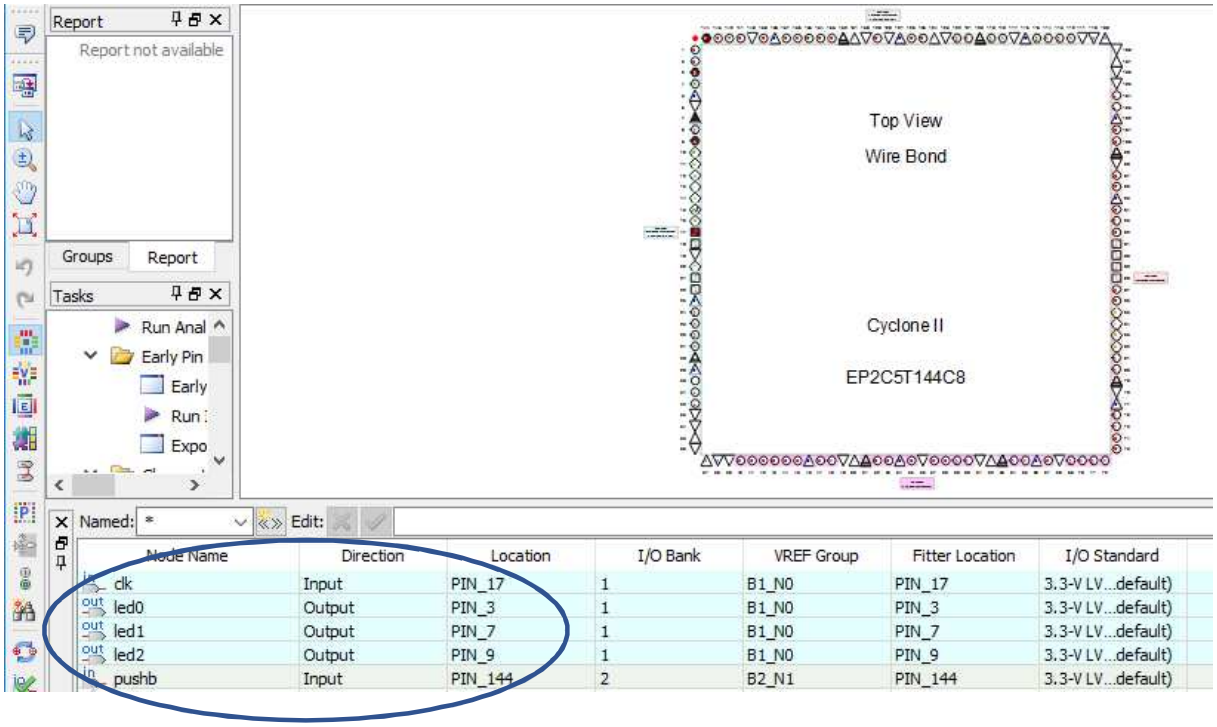


Fig. 19: Asignación de Pines para el Diseño PL_CLK.

La Fig. 20 muestra los recursos utilizados para la síntesis de este diseño, donde se puede notar que este diseño efectivamente necesita y utiliza recursos disponibles en la FPGA. Se pueden comparar los datos de esta Fig. 20 con los de la Fig. 18 anterior, que corresponde a un diseño mucho más sencillo.

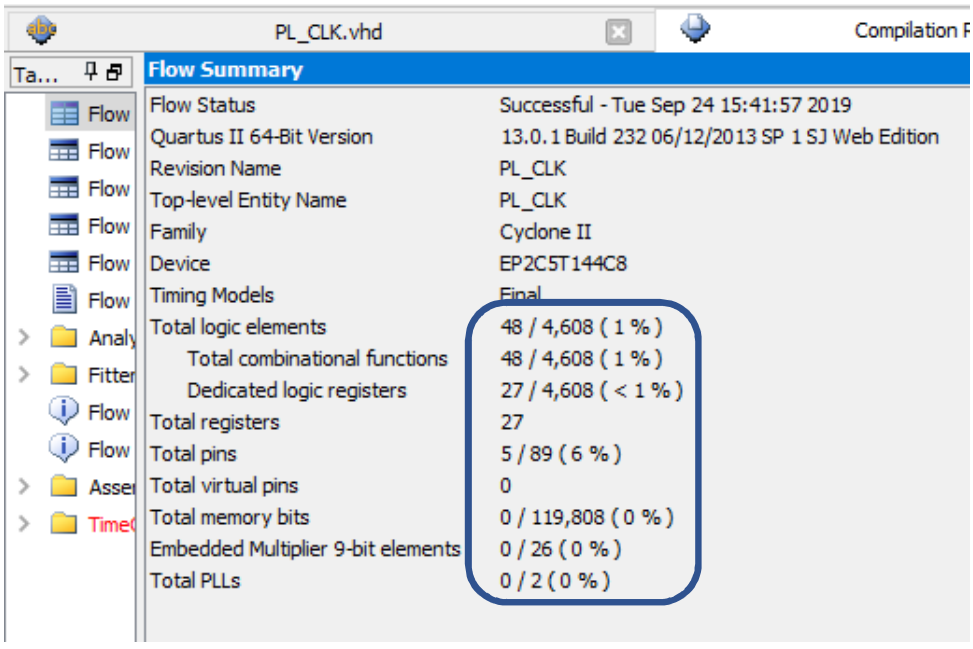


Figura 20: Recursos Utilizados para la Síntesis de PL_CLK.

# 7.- Conclusiones

Tal como se indicó en el inicio de este reporte, se deja documentado el proceso necesario para sintetizar un diseño especificado con VHDL en la FPGA Cyclone II integrada en la placa "EP2C5 Cyclone II Mini Board". La parte más sencilla de este proceso es la de instalación del IDE Quartus II específicamente provisto para diseño de hardware y síntesis en múltiples FPGA de (Intel) Altera. Como es usual, se aprovecharon los dispositivos integrados en la propia placa de desarrollo, como leds, botón pulsador y reloj de 50MHz.

Quizás como detalle extra, se mostró un problema que puede considerarse "clásico" en el diseño de hardware en general y en el diseño de hardware de/para sistemas embebidos, como es el problema de la "no previsibilidad" del valor de una señal cuando se deja un circuito abierto. Este caso específico, al ser tan sencillo es también sencilo para ser identificado. En diseños más complejos, la tarea se asemeja mucho a un "debug" de hardware, y en algunas situaciones la simulación del diseño no aporta mucha información relevante, suele ser necesario recurrir a la experimentación sobre el hardware mismo. Es importante notar que las simulaciones para los ASICs (Application Specific Integrated Circuits) las simulaciones son mucho más exhaustivas y las herramientas de simulación mucho más elaboradas (y costosas en todos los aspectos) y, por lo tanto, es de esperar que no sea necesario recurrir a un experimento o validación sobre el dispositivo mismo.

# Referencias


[CycloneII] Altera Corporation, Cyclone II Device Handbook, Volume 1, 2008
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyc2/cyc2_cii5v1.pdf.

[wiki] Cyclone II EP2C5 Mini Dev Board - blwiki
http://land-boards.com/blwiki/index.php?title=Cyclone_II_EP2C5_Mini_Dev_Board

[Heller] "Getting started with the EP2C5 Cyclone II Mini Board",
http://www.leonheller.com/FPGA/FPGA.html

[Intel1] Intel Corporation, "Altera DE1 Board"
https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de1-board.html

[Quartus] Intel Corporation, Download Center for FPGAs,
https://www.intel.com/content/www/us/en/programmable/downloads/download-center.html

[video1] FPGA Cyclone II pushled
https://youtu.be/EeuB5DaCk50

[video2] FPGA Cyclone II pushled - detalle
https://youtu.be/QG7qZYrMeDM

[video3] FPGA Cyclone II pushled - pullup
https://youtu.be/qLfV3C4LKr0

[video4] FPGA Cyclone II pushled + blinking
https://youtu.be/QZ5nOgdouNg