PRÁCTICA 4

Diseño RT -> FPGA usando Vivado.

Tutorial 2

1. Objetivos y Metodología

En este segundo tutorial se introducen algunas utilidades no cubiertas en el primero. En concreto IP-Integrator y el análisis temporal. Se vuelve a realizar un ejercicio de diseño guiado. El circuito es ahora secuencial y más complejo que el primero. El diseño es jerárquico. Posteriormente se proponen ejercicios con la herramienta Vivado.

Los archivos necesarios para la realización de estos tutoriales pueden descargarse desde enseñanza virtual: Practica3.zip.

2. TRABAJO PREVIO

P.1 Analice los tres módulos Verilog proporcionados: Registro.v, Calculo.v y Registro6.v. Describa la funcionalidad de cada uno de ellos.

P.2 Escriba una descripción Verilog estructural para el circuito de la figura. Los bloques Registro, Cálculo y Registro6 son los proporcionados.

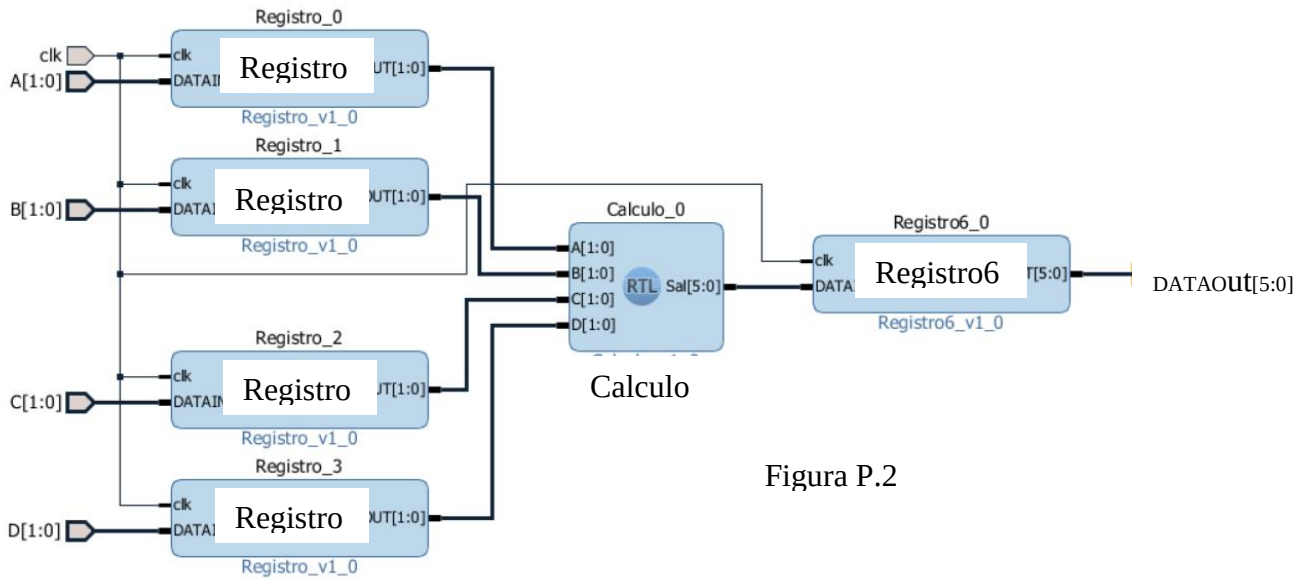


Figura P.2

P.3 Escriba una descripción no jerárquica (en un único ***module***) para el circuito anterior.

3. Flujo de diseño usando IP Integrator

IP Integrator es una potente utilidad incorporada en Vivado para facilitar el diseño de sistemas complejos a partir de bloques ya diseñados (IPs). Dispone de una interfaz gráfica para colocarlos e interconectarlos. Esto es, se "dibuja" un diagrama de bloques (***Block Design Diagram***) como punto de partida de nuestro diseño.

Muchos IPs se pueden configurar al instanciarlos (seleccionar por ejemplo propiedades o valores para parámetros). Para instanciar un IP en un diagrama de bloques, éste debe estar en el catálogo de IPs del proyecto. Este catálogo lo gestiona el diseñador.

La procedencia de los IPs puede ser:

- IPs de Xilinx. Una colección extensa de bloques de diseño distribuida con la propia herramienta. Estos IPs están disponibles cuando creamos un diagrama de bloques.
- IPs de otras procedencias. Estos tienen que añadirse al catálogo. En particular, el IP puede haber sido generado por el propio diseñador. Vivado también incluye utilidades para encapsular diseños en un IP y que puedan ser utilizados en otros.

La herramienta ayuda al diseñador en la interconexión de los bloques. Es capaz de interconectar automáticamente determinados bloques. Esto es muy útil para diseñar plataformas hardware basadas en microprocesadores. Este uso está fuera del alcance de esta asignatura.

En nuestros laboratorios haremos uso de su funcionalidad más simple, que permite incorporar componentes asociadas a descripciones HDL al diagrama de bloques. Estas descripciones HDL tienen que estar añadidas al proyecto como fuentes de diseño.

En este tutorial vamos a diseñar el circuito de la Figura P.2 a partir de descripciones Verilog de los registros y del bloque de cálculo utilizando ***IP Integrator***. Dichas descripciones Verilog están disponibles. Además, vamos a imponer restricciones temporales al diseño.

2.1 Creación del diagrama de bloques

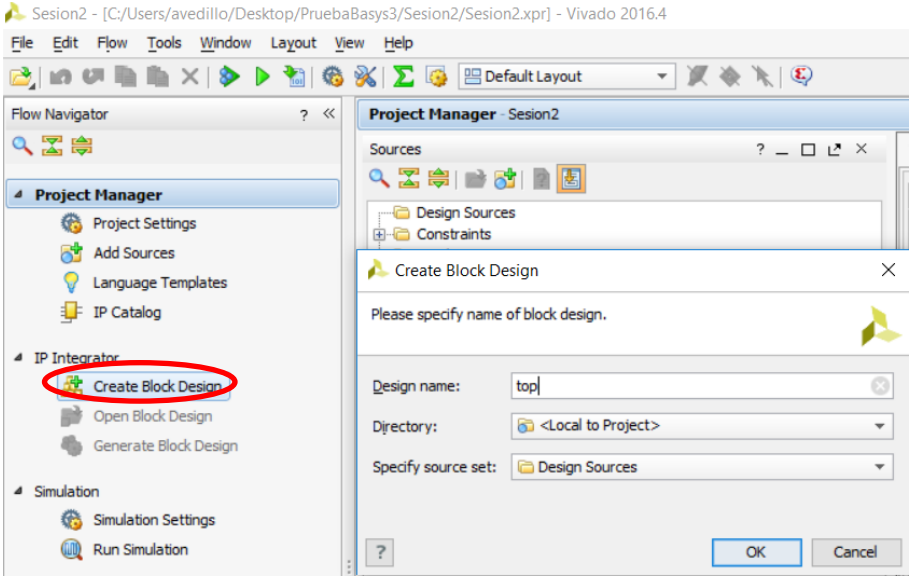
1.- Creamos un proyecto como en la Sesión 1.

2.- Añadimos como ***Design Sources*** las tres descripciones Verilog proporcionadas: Registro.v, Calculo.v y Registro6.v.

3..- Creamos un diagrama de bloques.

Seleccionamos en el ***Flow Navigator*** -> ***IP Integrator*** -> ***Create Block Design***

En la ventana que se abre indicamos un nombre (top) y dejamos la localización por defecto.



Pulsamos OK.

Se abre un lienzo de dibujo con una barra lateral de herramientas:

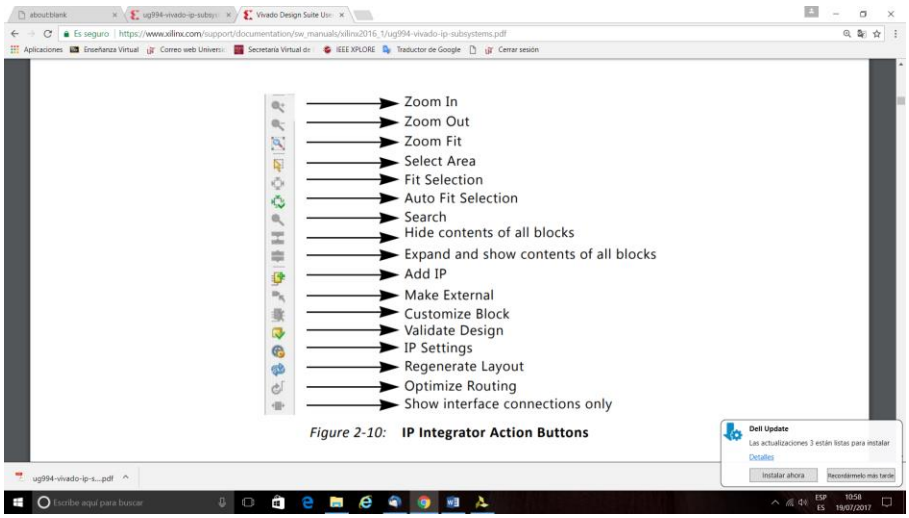
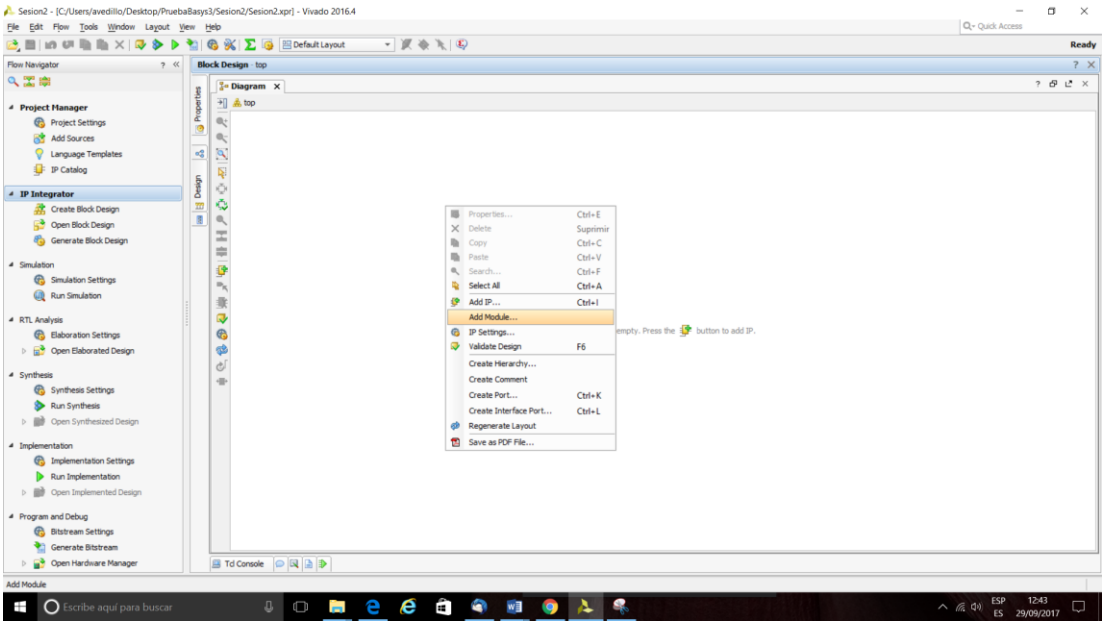


Figure 2-10: **IP Integrator Action Buttons**

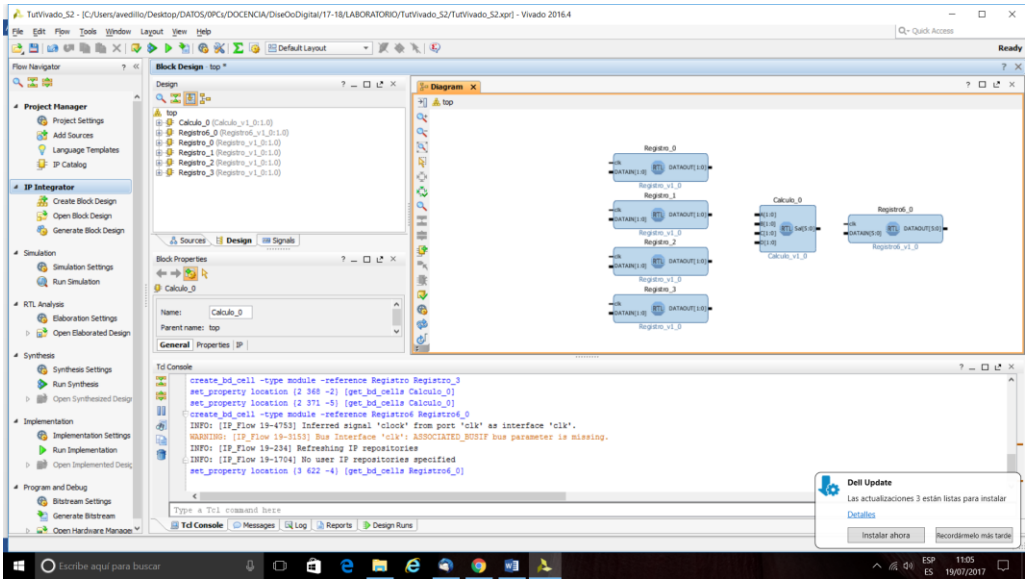
4.- Colocamos las componentes.

Sobre el lienzo de dibujo, pulsamos el botón derecho y seleccionamos ***Add Module***.



En la ventana que se abre aparecen las fuentes HDL del proyecto y se selecciona la que se quiere añadir. Pulsamos ***OK***.

Añadimos y colocamos todas las componentes de nuestro diseño. Para mover los bloques se pulsa el botón izquierdo y se arrastra.

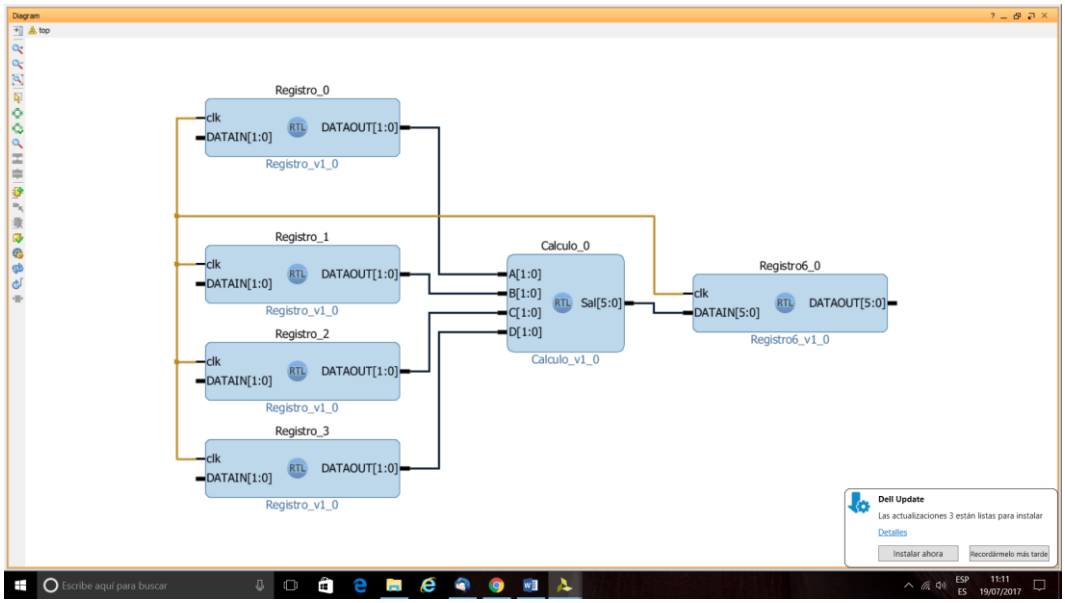


5.- Interconectamos las componentes.

Al picar en un terminal (entrada o salida) de un bloque, el cursor se convierte en un lápiz y podemos dibujar un cable. Para ello mantenemos pulsado el botón izquierdo y dibujamos la conexión hasta el bloque destino.

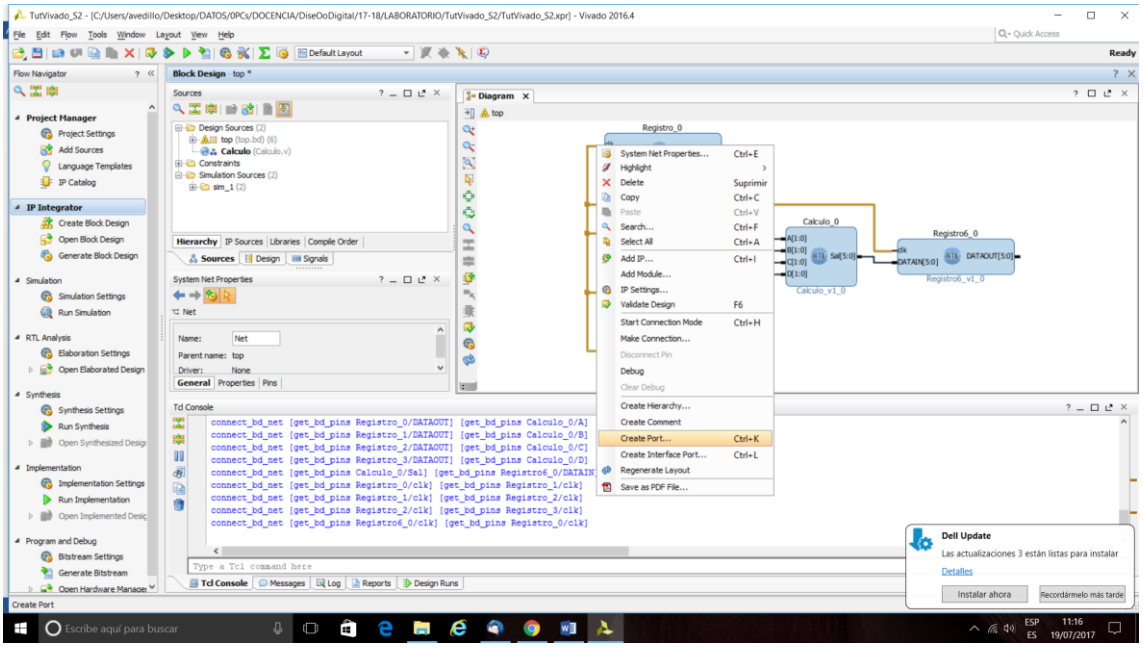
Conectamos las salidas de los registros de entrada a las entradas del bloque cálculo. Conectamos la salida del bloque cálculo a la entrada del registro de salida.

Conectamos los relojes de todos los registros.

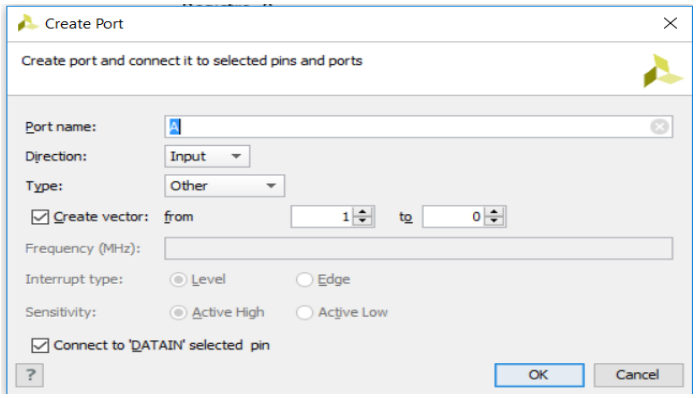


6.- Indicamos las entradas (*clk*, *A*, *B*, *C*, *D*) y la salida (*DATAOUT*) del diseño.

Para añadir un terminal, situamos el cursor sobre la señal que queremos hacer externa, pulsamos el botón derecho y seleccionamos ***Create port***.



Configuramos el terminal. Por ejemplo, le damos un nombre:



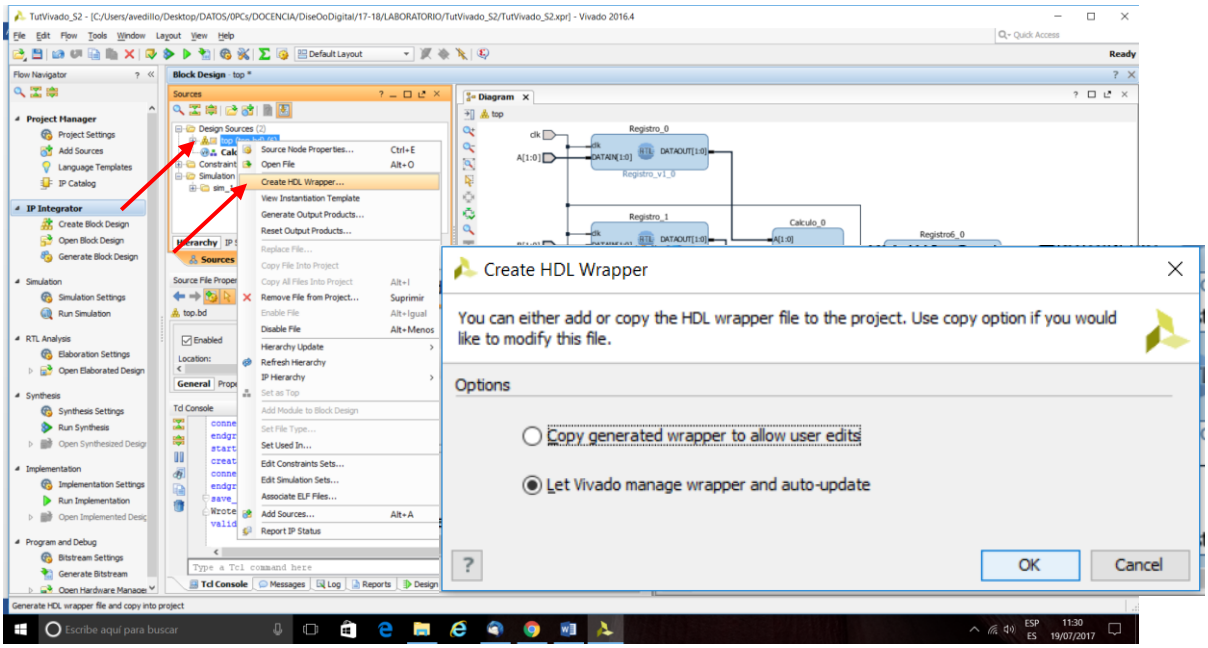
7.- Salvamos y validamos el diseño.

Para validar pulsamos en botón derecho sobre la zona de dibujo y seleccionamos ***Validate Design***.

8. Generaramos el HDL Wraper para el diagrama.

Una vez que el diagrama se ha completado y se ha validado, se genera una descripción HDL asociada.

Seleccionamos el diagrama (top.bd) en la ventana de fuentes, pulsamos el botón derecho y seleccionamos ***Create HDL Wrapper***. Dejamos la opción "*Let Vivado manage wrapper and auto-update*" seleccionada en la ventana que se abre y pulsamo OK.



En este punto podemos recorrer el flujo de diseño como se hizo en la Sesión 1.

2.2 Recorrido del flujo de diseño.

1.- Hacemos una simulación de comportamiento.

Añadimos el testbench (top_tb) proporcionado y simulamos. Analizamos los resultados.

2.- Añadimos una restricción temporal.

El archivo Sesion2.xdc proporcionado impone una restricción sobre la frecuencia de operación mínima. Debe funcionar a 100MHz.

Añadimos este archivo de restricciones al proyecto. Este es el contenido del archivo:

```
create_clock -name clk -period 10.000 [get_ports clk]
 set_input_delay -clock clk  0 [get_ports A[0]]
set_input_delay -clock clk  0 [get_ports A[1]]
set_input_delay -clock clk  0 [get_ports B[0]]
set_input_delay -clock clk  0 [get_ports B[1]]
set_input_delay -clock clk  0 [get_ports C[0]]
set_input_delay -clock clk  0 [get_ports C[1]]
set_input_delay -clock clk  0 [get_ports D[0]]
set_input_delay -clock clk  0 [get_ports D[1]]
set_output_delay -clock clk  0 [get_ports DATAOUT[0]]
set_output_delay -clock clk  0 [get_ports DATAOUT[1]]
set_output_delay -clock clk  0 [get_ports DATAOUT[2]]
set_output_delay -clock clk  0 [get_ports DATAOUT[3]]
set_output_delay -clock clk  0 [get_ports DATAOUT[4]]
set_output_delay -clock clk  0 [get_ports DATAOUT[5]]
```

3.- Sintetizamos y analizamos los resultados.

4.- Implementamos y analizamos los resultados.

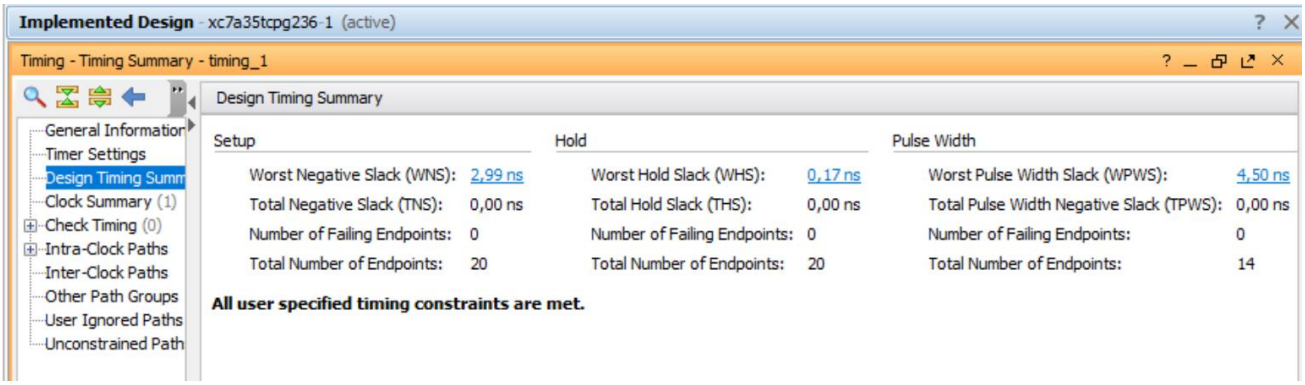
Vamos a generar también un informe sobre comportamiento temporal

Seleccionamos ***Implementation -> Timing Report Summary***.

Aceptamos las opciones por defecto para el análisis temporal que se va a realizar (pulsamos ***OK*** en la ventana que se abre).

Se abre el correspondiente informe en el área inferior de la pantalla. Se informa de que se cumplen las restricciones. Se observa que se han chequeado restricciones de *set_up* y de *hold*.

Este es el *Design Timing Summary*:



Setup:

Nos informa del margen (*slack*) del camino más lento. Esto es, incluso si este camino tuviese un retraso 2.99ns mayor, el circuito funcionaria a la frecuencia objetivo.

Ha analizado 20 endpoints porque necesita analizar caminos que terminan en un flip-flop (hay 14 flip-flops y por lo tanto 14 endpoints asociados) y caminos que terminan en salidas (hay 6 salidas)

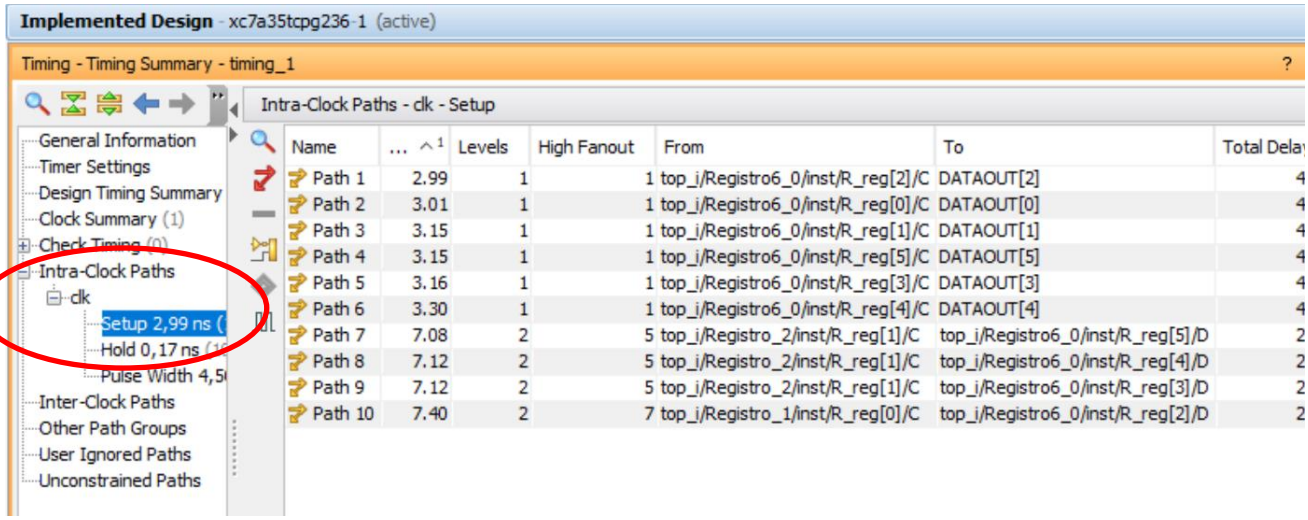
Hold:

Nos informa del slack del camino más rápido. Esto es, incluso si este camino tuviese un retraso 0.17ns menor, se estarían cumpliendo las condiciones de hold.

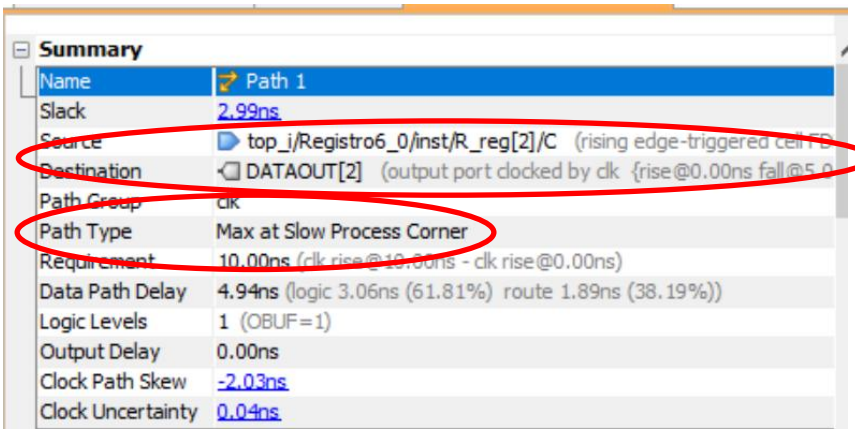
Pulse-width

Analiza el ancho del pulso de reloj que llega a cada flip-flop. Nos informa que este ancho es más que suficiente para el correcto funcionamiento del flip-flop. Tenemos un margen aún de 4.5ns.

Podemos ver cuáles son los 10 caminos con menores valores de slack para las condiciones de set_up:

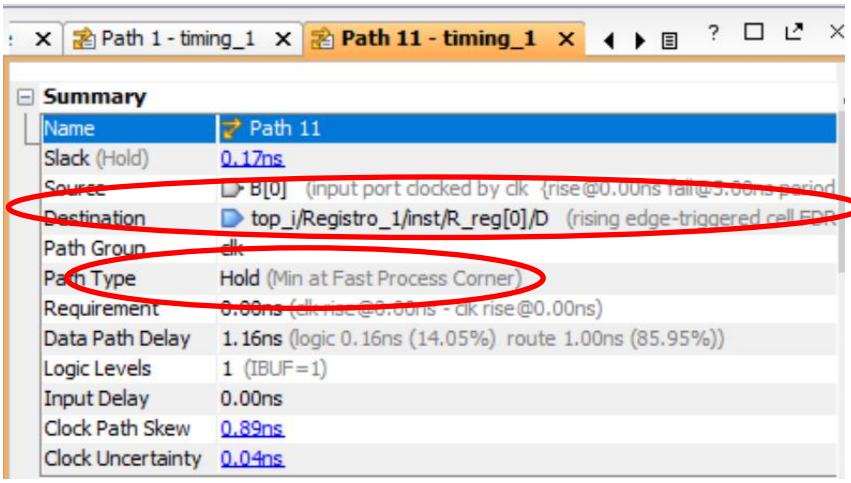


Podemos identificar cuáles son cada uno de esos caminos. Por ejemplo, seleccionando el más lento (path1):



Se observa que se trata de un camino que comienza en el bit 2 del registro de salida y termina en el bit 2 de la salida. Recuerde que en el circuito implementado se añaden buffers de salida entre el registro y el nodo que se denomina DATAOUT. Como corresponde al análisis de las condiciones de *set_up*, se usa los caminos con mayores retrasos (**Max**) y suponiendo que las puertas son lentas (***Slow***).

De la misma forma, para los caminos más rápidos y las condiciones de hold:



Se observa que se trata de un camino que comienza en el bit 0 de la entrada B y termina en el bit 0 del correspondiente registro de entrada. Recuerde que en el circuito implementado se añaden buffers de entrada entre las entradas y el registro en el que almacenan. Como corresponde al análisis de las condiciones de *hold*, se usa los caminos con mayores retrasos (**Min**) y suponiendo que las puertas son rápidas (**Fast**).

3. Ejercicios

Use un XDC con sólo la *constraint* asociada al reloj. Puede comentar las restantes líneas.

S2.1. Recorra el flujo de diseño completo usando la descripción correspondiente al ejercicio P.3 de este documento. Complete la primera fila de la tabla.

S2.2. Repita el diseño anterior para datos de 6 bits.

1. Recorra el flujo de diseño completo. Complete la segunda fila de la tabla.
2. Repita el proceso modificando una opción de síntesis.
    a. Abra ***Synthesis Settings*** y fije ***max_dsp*** a 0 (por defecto vale -1) y pulse ***OK***
    b. Seleccione ***yes*** en la ventana ***New Run*** que se abre para conservar los resultados anteriores.

Sistema	**Flip-Flops**	**LUTs**	**DSPs**	**Set up Slack Post implementation**
2 bits				
6 bits				
6 bits *max_dsp* = 0				

S2.3 Para el primero de los diseños de 6 bits identifique el camino con menor margen de setup y el camino con menor margen de hold. Debe indicar su origen y su destino.