

# ***PRÁCTICA 1***

***Diseño RT -> FPGA usando Vivado.***

## ***Tutorial 1***

## 1. Objetivos y Metodología

El objetivo de esta primera sesión de laboratorio es adquirir una visión general de la herramienta de diseño Vivado Design Suite de Xilinx y recorrer el flujo de diseño RT -> FPGA completo utilizándola. Se realizará un tutorial guiados de Vivado. Esto es, un ejemplo de diseño realizados por el profesor y, simultáneamente, por los alumnos en sus puestos. Posteriormente se proponen ejercicios con la herramienta Vivado.

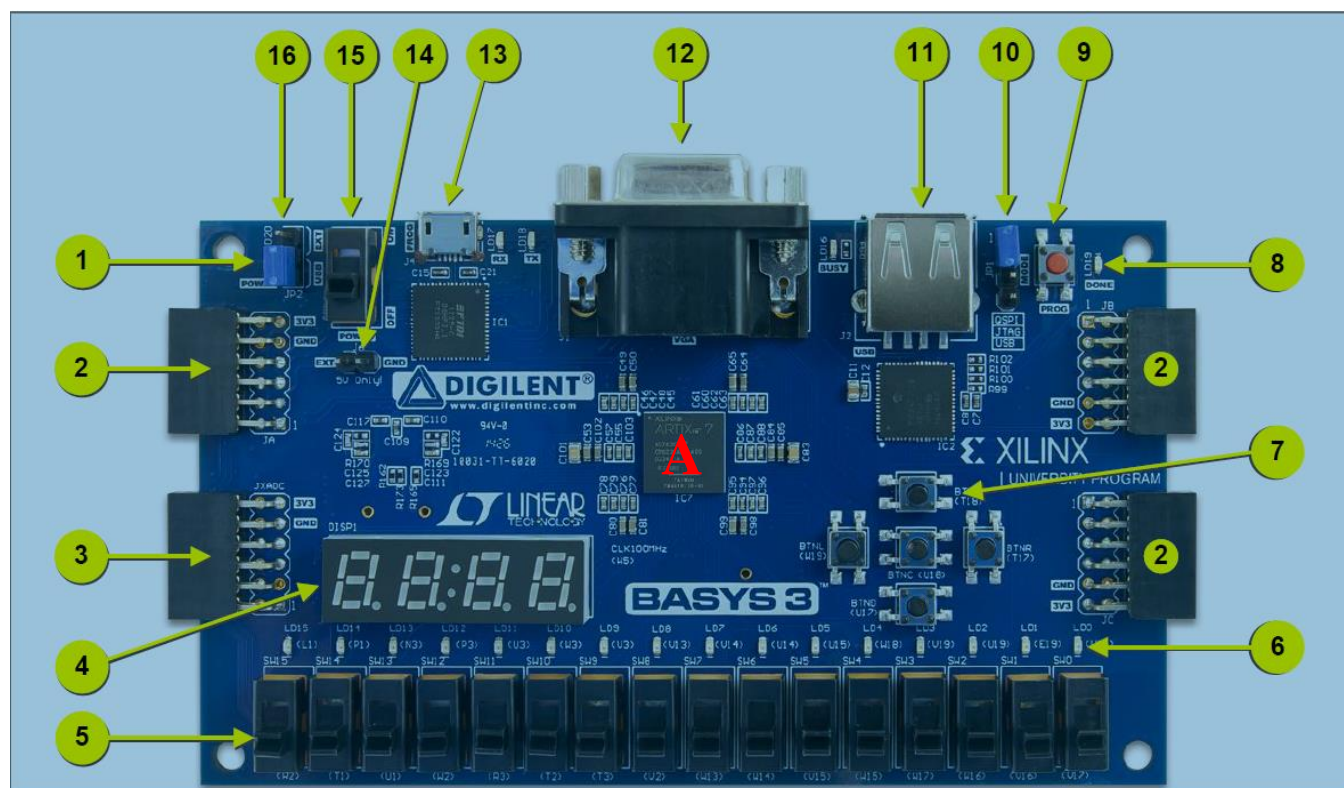
Este documento no pretende substituir a la extensa documentación existente sobre el uso y manejo de la herramienta (*Start Guides*, *Tutorials* y *User Guides*) y fácilmente accesibles desde el menú **Help** del propio entorno de diseño. Se trata de un resumen de los tutoriales que se desarrollan en clase. Esta documentación se ha preparado para la versión 2016.4 instalada en el laboratorio.

Los archivos necesarios para la realización de estos tutoriales pueden descargarse desde enseñanza virtual: Practica1.zip.

## 2. Placa de desarrollo Basys3

Para la realización física de los sistemas digitales diseñados en las distintas prácticas de la asignatura DSD se utiliza la placa de desarrollo de sistemas digitales **Basys3** de Digilent que contiene un FPGA de Xilinx.

### 2.1. Descripción general



Además del CI correspondiente al FPGA (A), dispone de un oscilador de 100 MHz (conectado a un determinado pin del FPGA), un chip de memoria (IC6 en el reverso de la placa, 32Mb flash no volátil,

conectada al FPGA mediante un bus SPI dedicado), otros CIs (reguladores, USB\_UART bridge, ... ) y diferentes puertos I/O y periféricos.

Los puertos y periféricos disponibles incluyen:

15	Encendido/apagado de la placa
16	<i>Jumper</i> de selección de fuente de alimentación. Las opciones son: USB, externa
14	Alimentación externa. Debe de ser de 5V
13	Puerto USB de alimentación, programación (JTAG) y comunicación serie FPGA-PC
1	Led que indica que la placa está alimentada
10	<i>Jumper</i> de selección de modo de programación. Las opciones son: JTAG, memoria flash y USB
4	4 <i>displays</i> siete segmentos. Visualizar salidas del diseño
5	16 conmutadores ( <i>switches</i> ). Se usan para controlar entradas de los diseños
6	16 leds. Salidas del diseño
7	5 pulsadores. Entradas del diseño. Pulsados generan un pulso de 1 lógico
9	Botón para resetear la programación del FPGA
11	Conector USB. Puede usarse para conectar un <i>pendrive</i> con un archivo de configuración para el FPGA, un ratón o un teclado
12	Conector VGA
2, 3	Conectores Pmod. Para expandir la Basys3 (conectar otros componentes)

Puede encontrarse una descripción más detallada en el documento **Manual Referencia\_Basys3** disponible en enseñanza virtual.

## 2.2. FPGA CX7A35T-1CPG236C

Este FPGA es una **Artix-7**. La familia Artix7 usa una tecnología de 28nm. La siguiente tabla muestra los recursos disponibles en distintos dispositivos de esta familia. En particular, en la que usaremos.

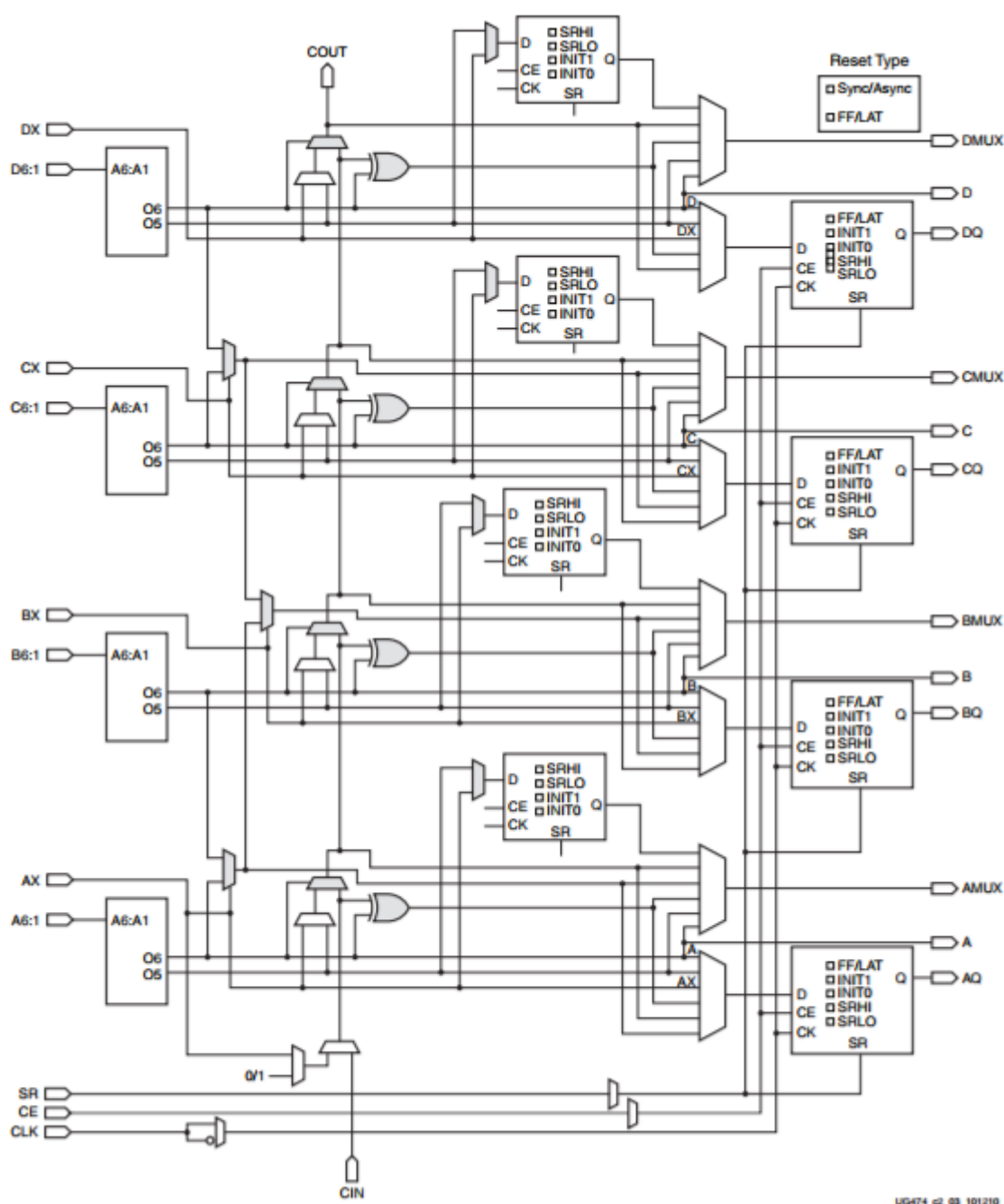
Table 1: XA Artix-7 FPGA Device-Feature Table

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices <sup>(2)</sup>	Block RAM Blocks <sup>(3)</sup>			CMTs <sup>(4)</sup>	PCIe <sup>(5)</sup>	GTPs	XADC Blocks	Total I/O Banks <sup>(6)</sup>	Max User I/O <sup>(7)</sup>
		Slices <sup>(1)</sup>	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XA7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XA7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	210
XA7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XA7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	210
XA7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	210
XA7A75T	75,520	11,800	892	180	210	105	3,780	6	1	4	1	6	285
XA7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	4	1	6	285

- Cada **CLB** de la familia Artix7 contiene 2 **Slices**. Cada Slice 4 LUTs de 6 entradas, 8 biestables y alguna lógica extra (multiplexores, lógica de acarreo). La figura muestra el diagrama lógico de un slice básico (SLICEL).

- Algunos slices (SLICEM) pueden usar sus LUTs para implementar memorias (es lo que se denomina memoria distribuida) o registros de desplazamiento.
- Los bloques DSPs permiten implementar funciones aritméticas de forma más eficiente que usando los CLBs.
- El tamaño de los bloques de memoria es de 36Kb pero puede usarse como 2 de 18Kb.
- CMTs. Bloques relacionados con la generación y distribución de señales de reloj.

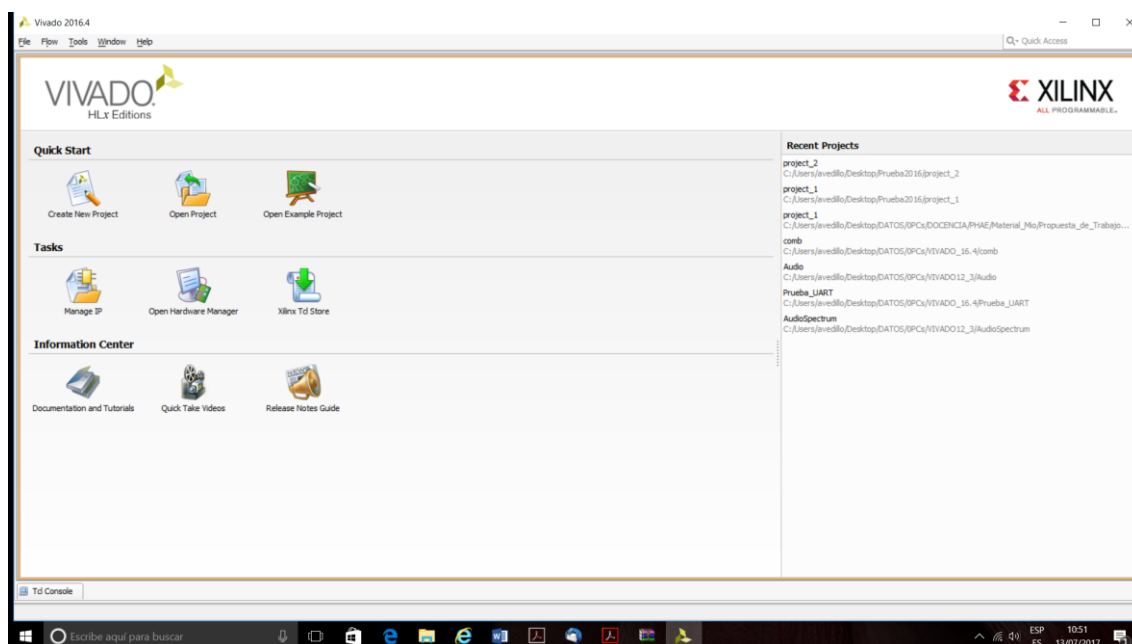
Puede encontrar más información en enseñanza virtual.



### 3. Flujo de diseño Verilog -> FPGA con Vivado Design Suite

#### 3.1.1. El entorno de diseño y los proyectos

Abrimos el entorno de diseño picando dos veces en el icono correspondiente:

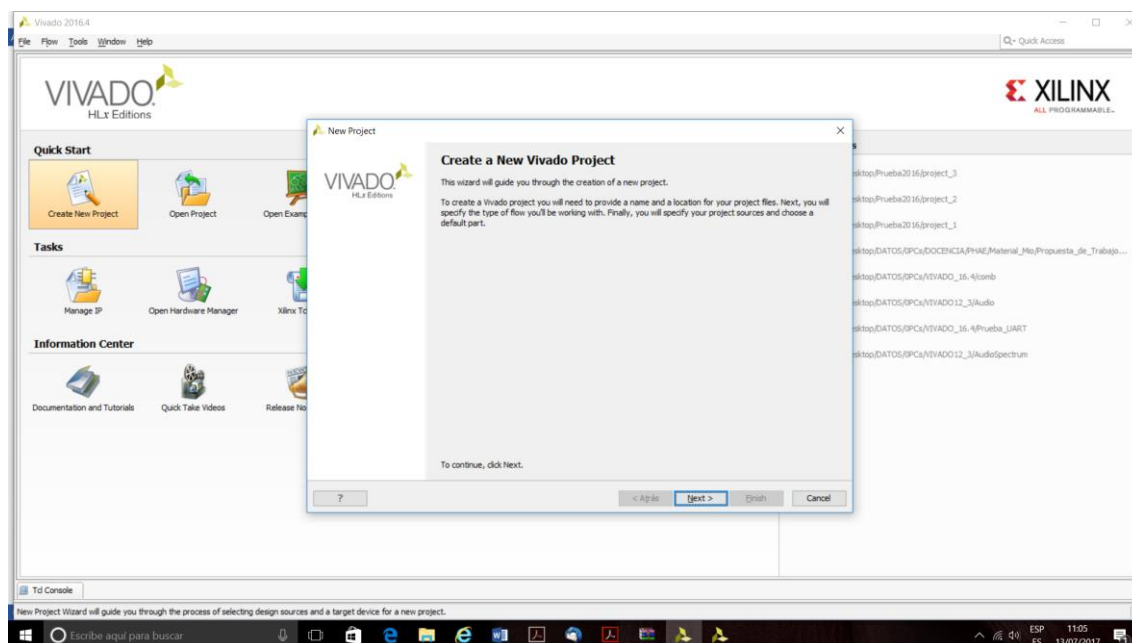


Desde el área **Quick Start** podemos crear un proyecto nuevo, abrir un proyecto existente o crear un proyecto utilizando alguno de las plantillas disponibles.

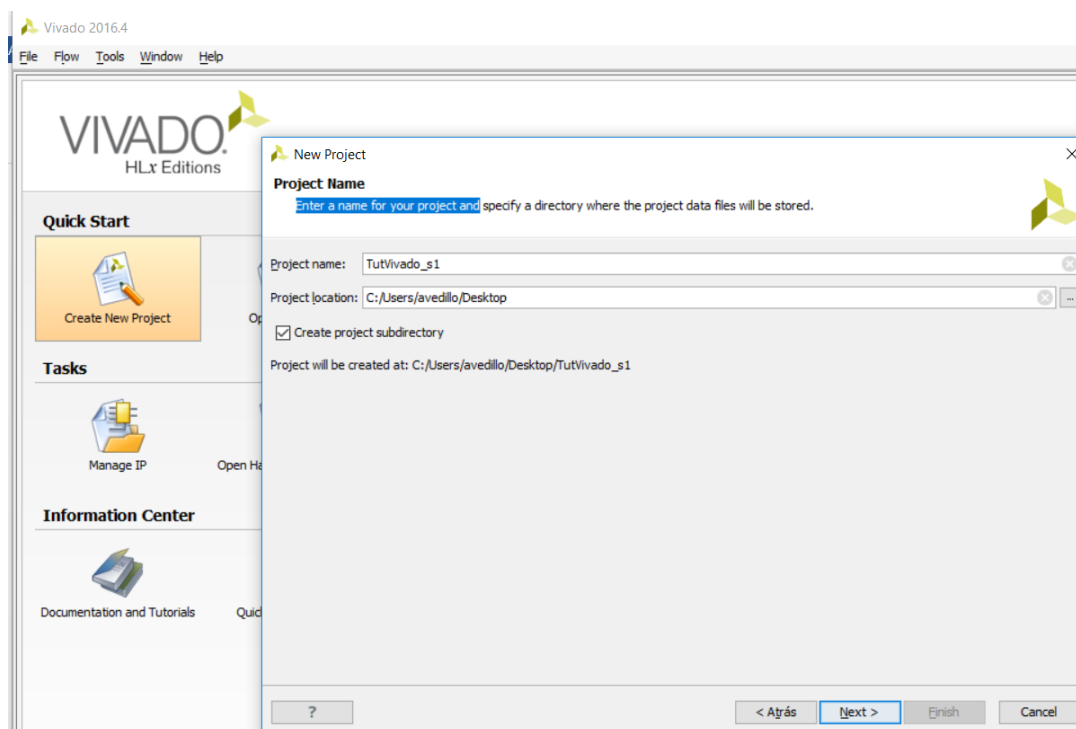
En este tutorial creamos un nuevo proyecto desde cero.

Seleccionamos **Create New Project**.

Picamos **Next** en el asistente para la creación de proyectos que se abre:

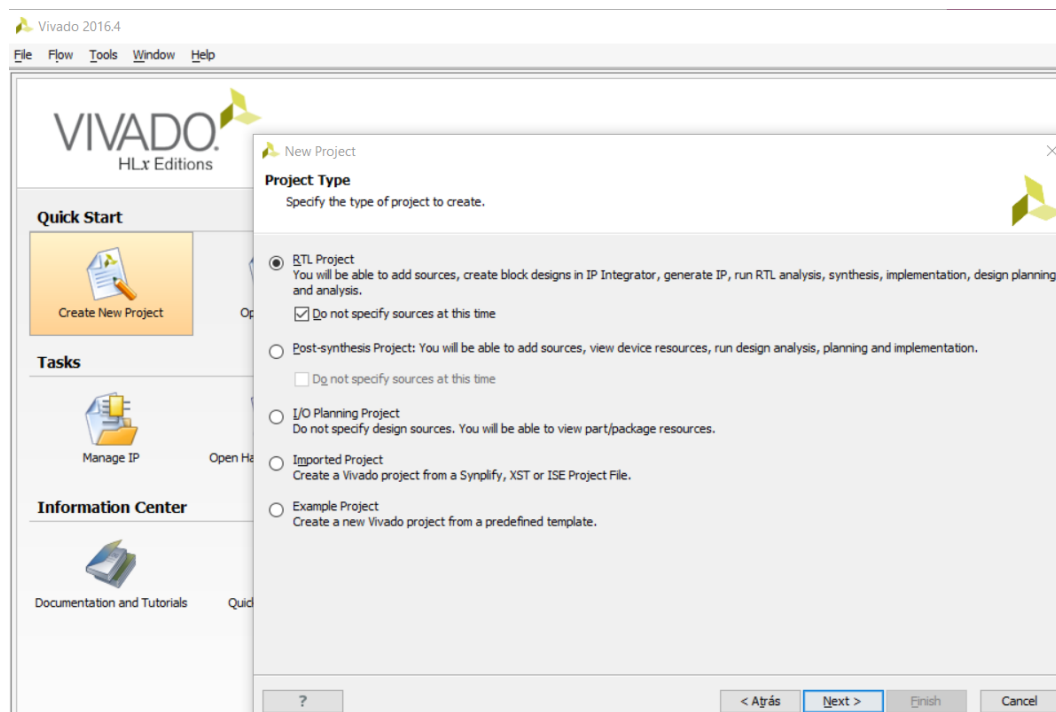


Nos Solicitará un nombre y una localización para el proyecto:



Completamos y pulsamos **Next**.

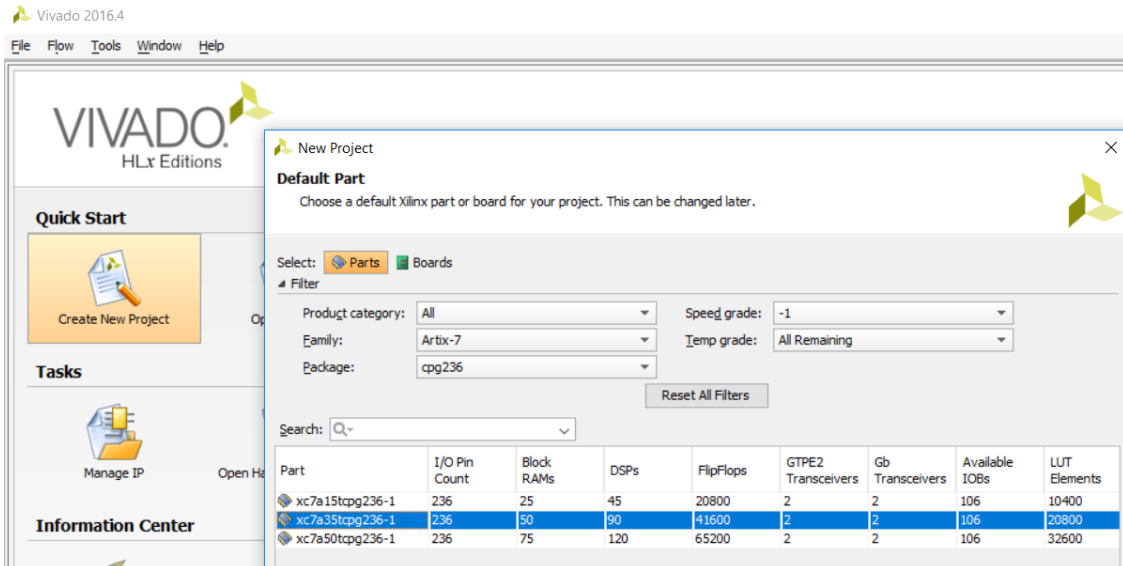
No solicita el tipo de proyecto. Seleccionamos **RTL Project** y clicamos **Do not specify sources at this time**:



Pulsar **Next**.

Nos solicita el FPGA (**part**) o la placa (**board**) en la que vamos a realizar el diseño. Seleccionamos **part** y el dispositivo que tiene la placa Basys3 (XC7A35T-1CPG236). La Identificamos utilizando los desplegables.

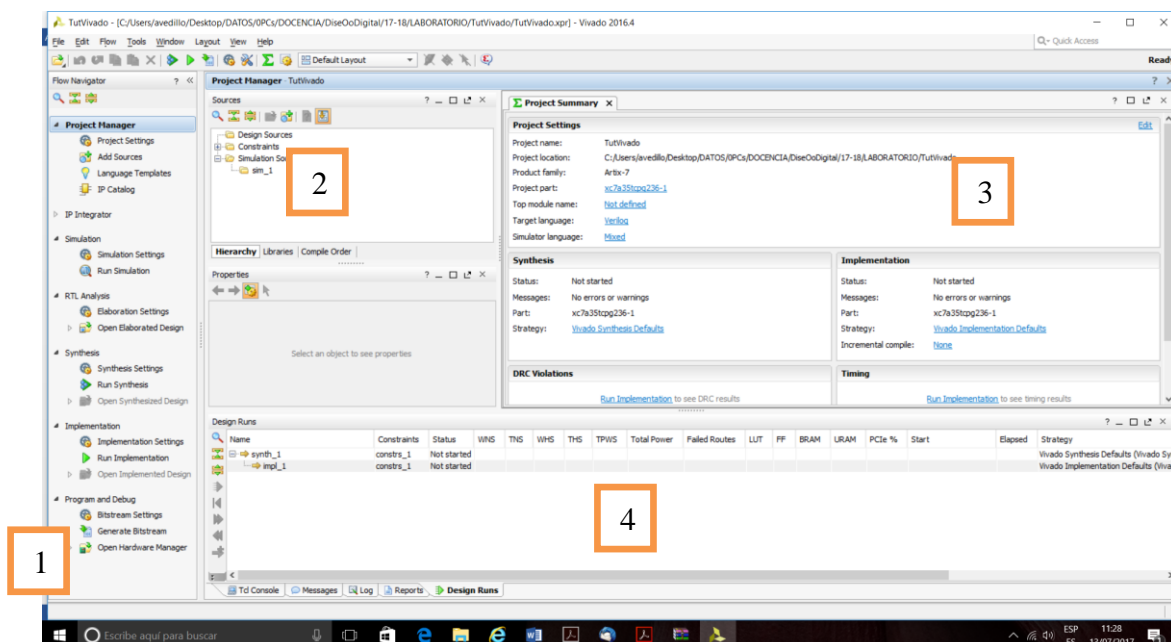
Family: Artix-7  
Device: **xc7a35t**  
Package: **cpg236**  
Speed Grade: **-1**



Pulsamos **Next**.

Pulsamos **Finish**.

Se abre la interfase para la gestión de diseños. Permite recorrer el flujo de diseño y analizar los resultados obtenidos.





Distinguimos 4 áreas fundamentales diferenciadas:

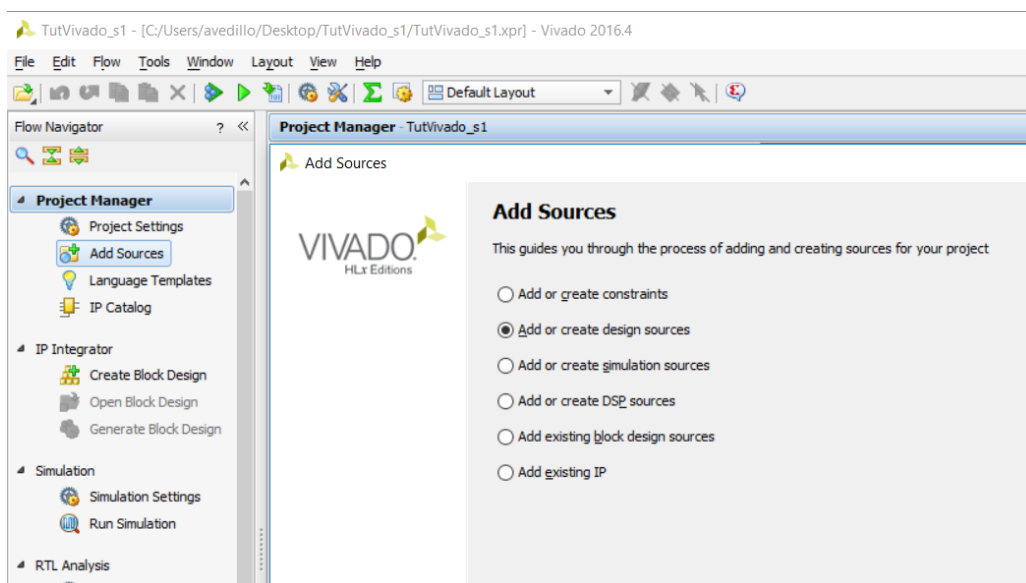
- 1.- **Flow Navigator**. Contiene el **Project manager** y las etapas del flujo de diseño (básicamente simulación, síntesis, implementación y programación del dispositivo).
- 2.- **Fuentes del proyecto** (parte del **Project manager**). En la ventana **Hierarchy** contiene las fuentes del diseño organizadas por “tipos”: descripciones de diseños (**Design Sources**), **testbenches** (**Simulation Sources**) y otras que serán necesarias en distintos pasos del proceso de diseño (**Constraints** u otros).

Existen subcarpetas para organizar las fuentes de simulación y los constraints.

- 3.- **Área de visualización**. Visualización de los ficheros fuentes del proyecto y determinados resultados.
- 4.- Comunicación con la herramienta.

### 3.2. Adición de fuentes al proyecto

Seleccionamos **Add Sources** en el **Project Manager** de la ventana **Flow Navigator** o en el Menu **File**.



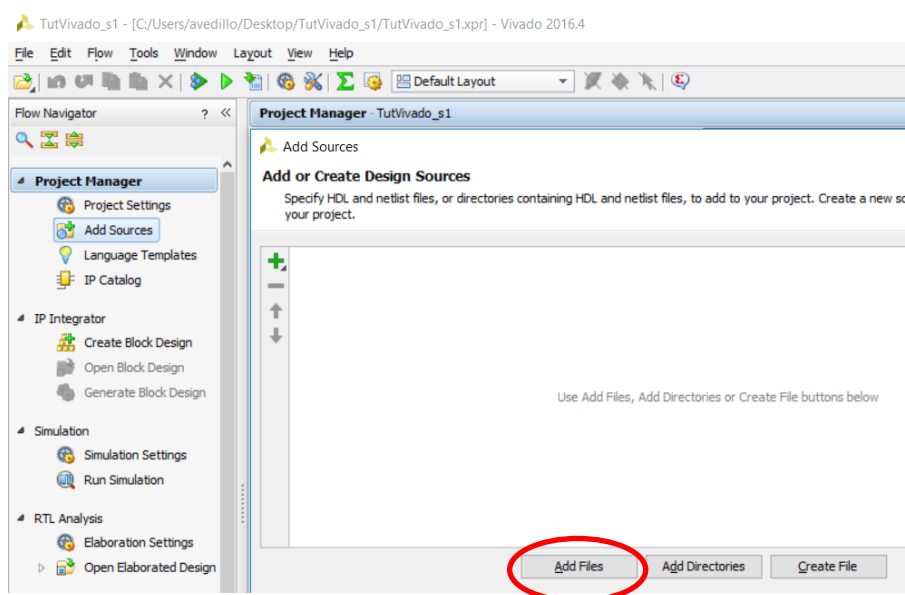
Seleccionamos el tipo de fuente que queremos añadir o crear.

#### Fuentes existentes (añadir)

Comenzamos por la fuente que contiene la descripción HDL del circuito que vamos a implementar (**Add or create design sources**). Pulsamos **Next**.

En este caso, la descripción ya existe (se ha proporcionado el archivo VivadoTut.v) y pulsamos **Add file**.





Se abre una ventana para seleccionar el archivo. Una vez seleccionado pulsamos **OK(open)**.

Pulsamos **Finish**.

Observamos que el module *TutVivado* aparece ya en **Design Sources**.

### Fuentes nuevas(crear)

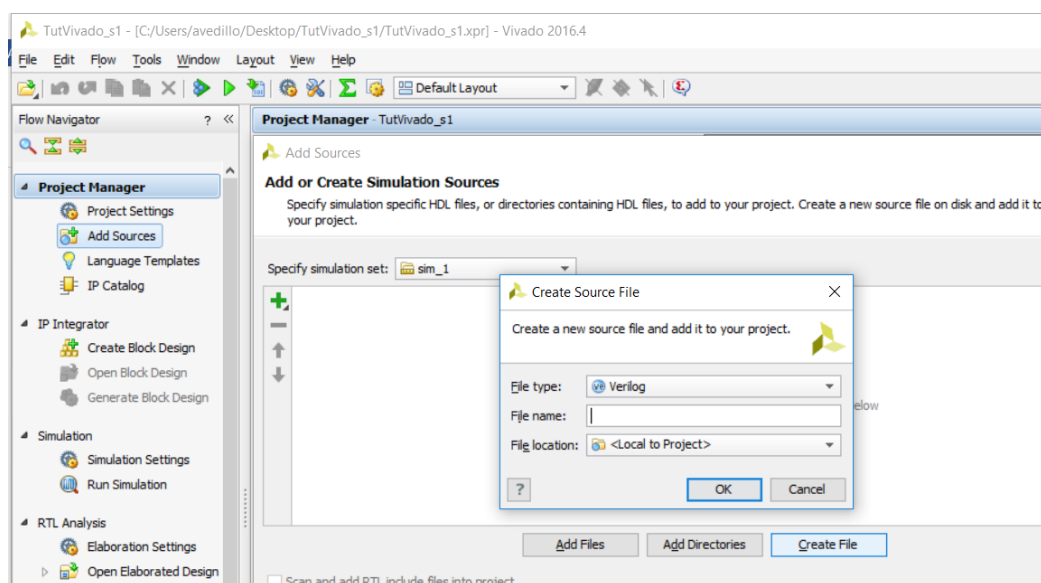
Vamos a crear una fuente nueva, de tipo *testbench*, para poder simular el diseño.

Seleccionamos **Add Sources** en el **Project Manager** de la ventana **Flow Navigator** (o en el Menu **File**), indicando que queremos una fuente para simulación (**Add or Create Simulation Sources**).

Pulsamos **Next** y seleccionamos **Create File** en la ventana que se abre.

Nos pide un tipo, un nombre y una localización para el archivo que vamos a crear.

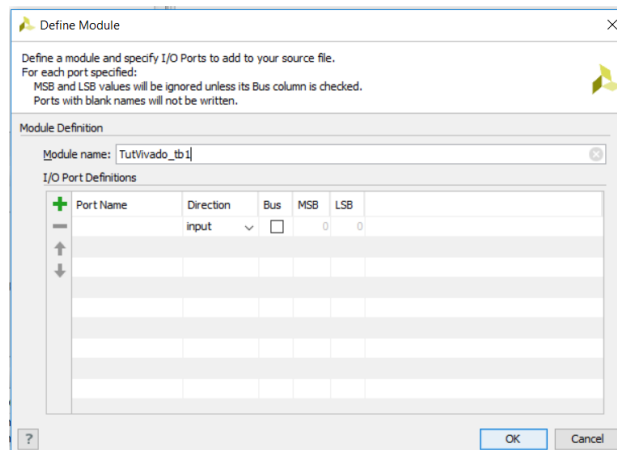
En tipo elegimos Verilog (HDL en el que vamos a escribir el testbench).



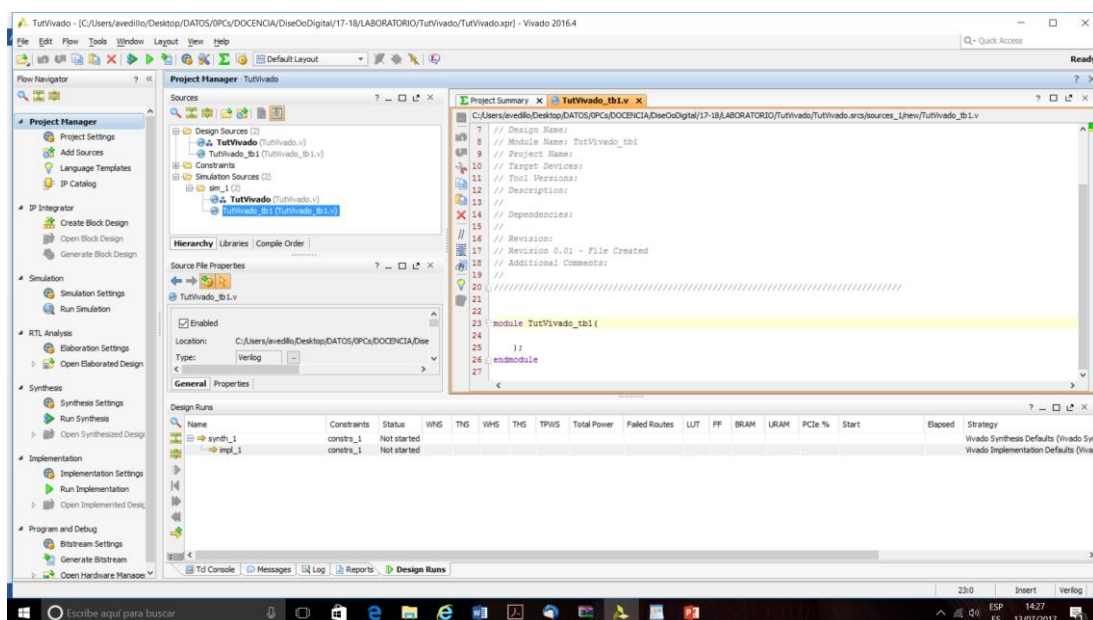
Damos un nombre y Pulsamos **OK**.

Pulsamos **Finish**.

Pulsamos **OK** en la ventana **Define Module**.



El archivo creado aparece bajo **Simulation Sources** en la ventana de **Sources**. Picamos dos veces para abrirlo en el área de visualización.



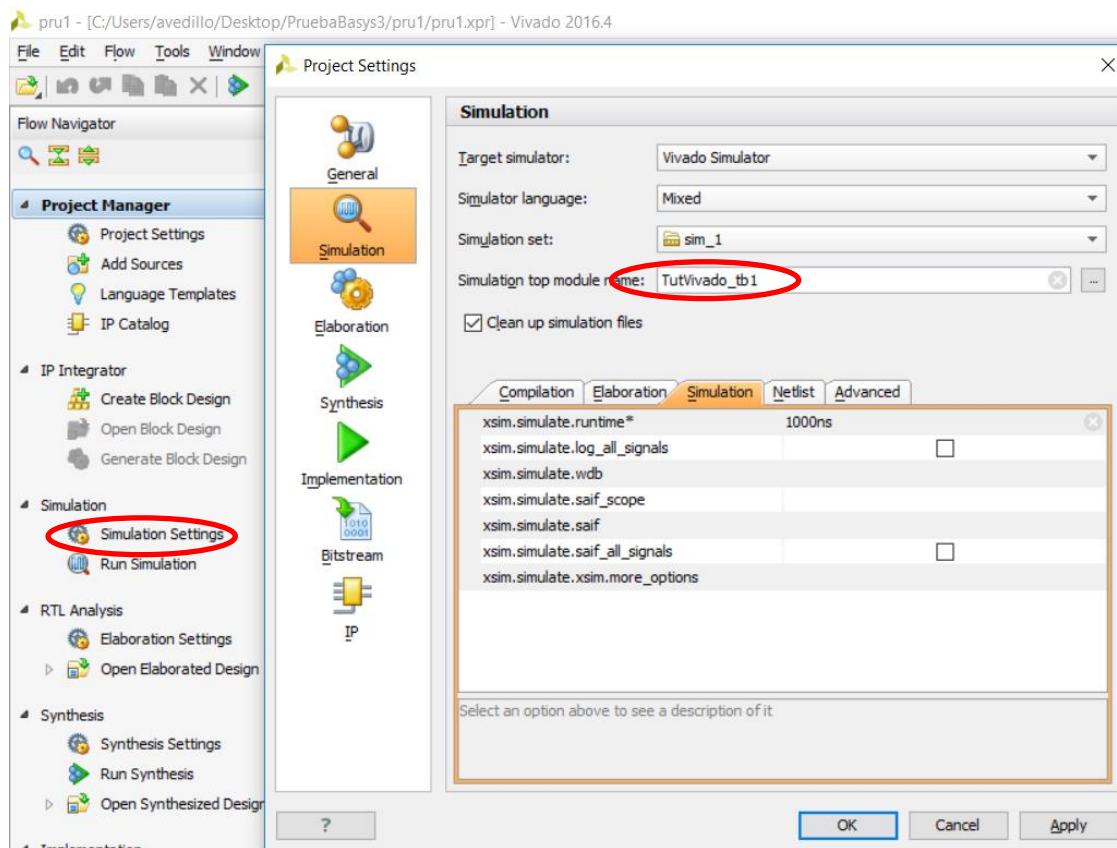
Se abre un editor de HDL (reconoce palabras claves por ejemplo). En este primer ejemplo no vamos a desarrollar el testbench, sino que vamos a copiar y pegar el contenido del archivo **TutVivado\_tb1.txt** descargado y a salvarlo (**Ctrl+S** o botón derecho **Save File**).

Observamos que se ha entendido la jerarquía de diseño creada al haberse instanciado el module **TutVivado** en el module **TutVivado\_tb1**.

### 3.3. Simulación de Comportamiento (Behavioural Simulation)

En el entorno Vivado se realizan distintos tipos de simulación (simulaciones en distintos puntos del flujo de diseño). Entre ellas: simulaciones comportamiento (**behavioural**, se simula el código HDL que describe el circuito) o simulación **post place&route** (se simula un modelo del circuito generado tras la síntesis y la implementación). Comenzamos por hacer una simulación de comportamiento para validar la funcionalidad de la descripción HDL.

Pulsamos **Simulation Settings** en el **Flow Navigator** para seleccionar el testbench que vamos a simular y, en su caso, controlar otras opciones de simulación:



Pulsamos **OK**.

Seleccionamos **Run Simulation -> Simulate Behavioural Model**. Esto lanza la simulación y una vez completada, se abre una ventana de simulación en la que vemos:

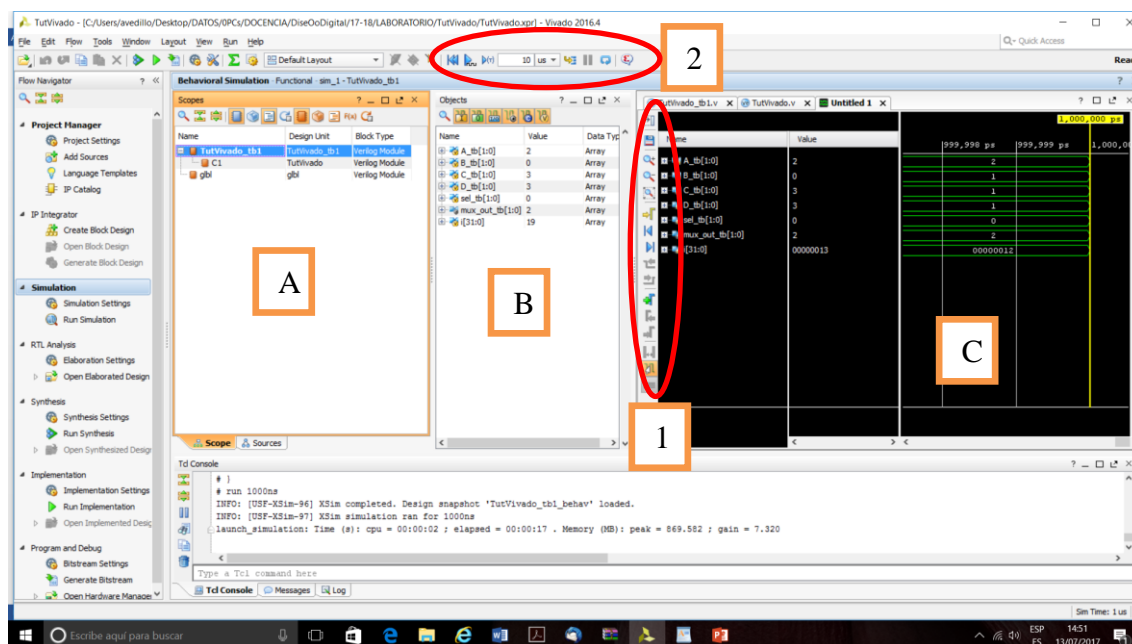
A.- Ventana **Scope** (A en la siguiente figura) que nos permite movernos por la jerarquía de diseño. Por defecto está seleccionado el *module* correspondiente al *testbench*.

B.- Ventana **Objects** (B) que tiene las señales asociadas al módulo que este seleccionado en **Scope**. Las señales en este caso son todas las declaradas en el module *TutVivado\_tb1*.

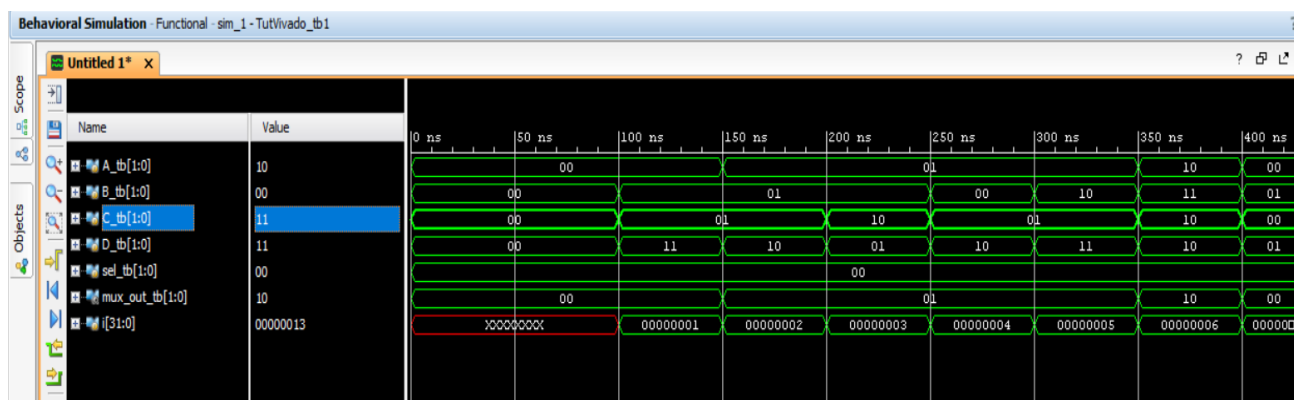
C. Ventana de visualización de formas de onda. Inicialmente muestra las señales del testbench. Como veremos, es posible representar también señales interna si existiesen.

Con la barra de herramientas lateral (**1 en la siguiente figura**)) se puede, entre otras cosas, cambiar el Zoom. En particular indicar que se muestre todo el tiempo simulado. Por defecto se simulan 1000ns. Seleccionando las señales en el visualizador se puede cambiar el tipo de representación (binario, hexadecimal,...) o el color.

Se puede simular más tiempo o reiniciar la simulación desde la barra de herramientas superior (**2**).



La respuesta del circuito se analiza para determinar si se corresponde con lo esperado.



Se puede cambiar el formato elegido para la representación de las señales:

Seleccionamos la señal o señales, pulsamos el botón derecho y seleccionamos **Radix**. Elegimos la opción deseada (*Binary, Hexadecimal, Octal, ...*)

De la misma forma se pueden configurar los colores y otras características.

Se puede salvar (segundo icono en la barra lateral de la ventana de visualización) la configuración del visualizador. Puede ser útil si se ha personalizado, para evitar repetir esta tarea en futuras simulaciones.

Cerramos la simulación pulsando X en la barra del **Behavioral Simulation** View. Pulsamos Ok y salvamos o descartamos la configuración del visualizador de formas de onda.

### 3..4. Restricciones

Es posible imponer restricciones (**User Constraints**) de distinto tipo a un diseño. Estas restricciones se guardan en un archivo (con extensión **.xdc**). Las restricciones se pueden añadir a un diseño en distintos puntos del flujo y usando distintos mecanismos:

- Se puede añadir a un proyecto un archivo **.xdc** ya existente
- Se puede crear un archivo **.xdc** nuevo y editarlo en un editor de texto (manual)
- Se puede abrir un diseño (sintetizado o implementado) y usar interfaces del entorno que simplifican la creación y modificación de las constraints. En este caso es necesario guardar las restricciones nuevas /modificadas en un archivo **.xdc** antes de poder sintetizar o implementar.

Existen dos tipos de restricciones:

**Temporales.** Imponen restricciones sobre el comportamiento temporal del circuito:

- Las especificaciones de un diseño incluyen especificaciones sobre su comportamiento temporal: e.j. retrasos para un circuito combinacional, frecuencia de operación para un circuito secuencial.
- El circuito resultante (tras la síntesis y la implementación) puede ser funcionalmente correcto, pero no cumplir con las especificaciones temporales.
- Al añadir las constraints temporales estamos informando a la herramienta de diseño de esos aspectos temporales de la especificación:
  - Al sintetizar e implementar se tienen en cuenta
  - Se facilita la validación temporal. Esto es, el chequeo de que el diseño resultante cumple con las especificaciones temporales, y que no presenta problemas de operación relacionados con los retrasos de las componentes e interconexiones que conforman el circuito.

**Físicas.** Controlan la localización de distintas componentes del circuito en el FPGA. En particular, suele ser necesario imponer restricciones sobre qué terminal físico asociamos a cada señal de entrada o salida (restricciones impuestas por la placa). Para cada placa hay un archivo "master" que simplifica esta tarea.

En este primer tutorial sólo vamos a imponer restricciones físicas. Disponemos del archivo máster para la placa (**Basys3\_Master.xdc**). Lo vamos a añadir al proyecto y lo vamos a editar para adecuarlo a nuestro diseño.

Para añadirlo seguimos el procedimiento ya descrito, indicando ahora **Add or create constraints**. Una vez añadido, lo abrimos para editarlo:

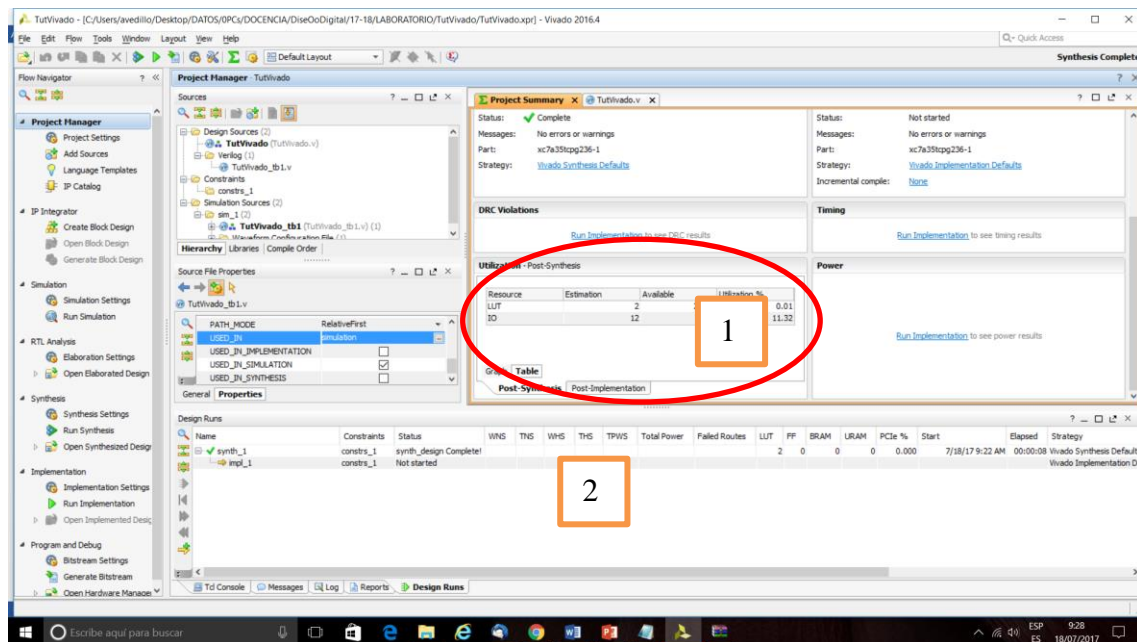
- Descomentamos los pines que vamos a usar en el diseño. Por ejemplo, vamos a usar *switches* para las entradas del diseño (2 para la señal de selección (*sel*) y 2 para cada uno de los datos de entrada, *A*, *B*, *C*, *D*. Para observar las salidas (*mux\_out*) usamos 2 leds.
- Renombramos los puertos utilizados (después de **get\_ports**) usando el los nombres de señal de nuestro diseño.

### 3..5. Síntesis

Una vez validada la funcionalidad del código HDL, el siguiente paso es sintetizarlo.

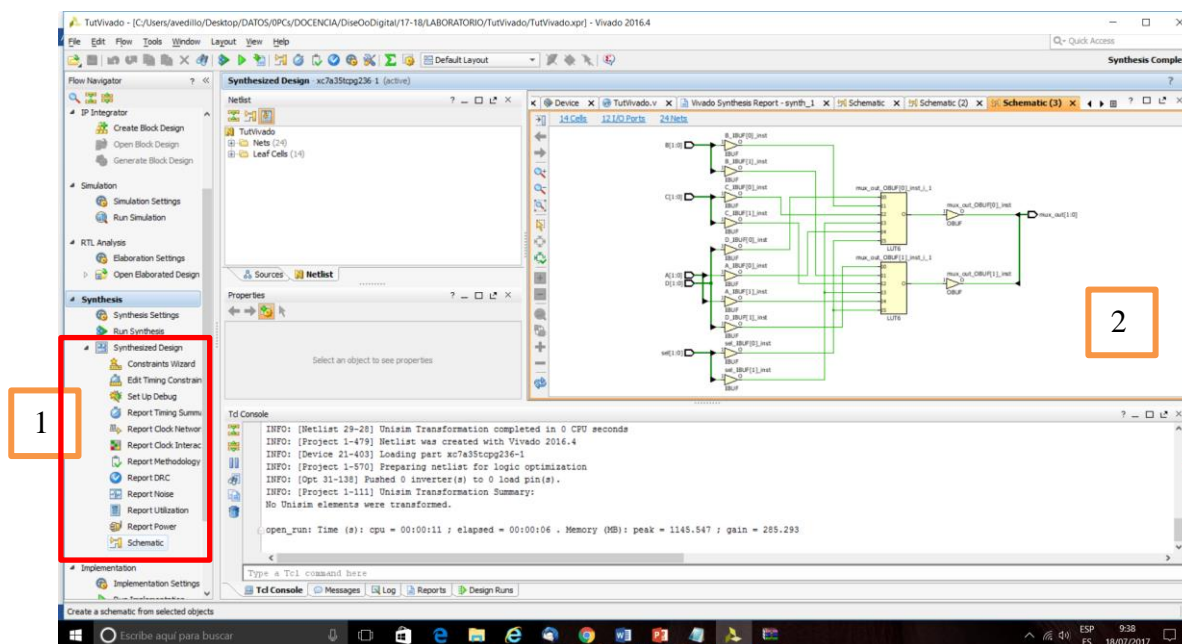
Pulsamos **Run Synthesis** en el área del **Flow Navigator**. Pulsamos **OK** en la ventana que se abre.

Una vez sintetizado el circuito, pulsamos cancel en la ventana que se abre. Está ventana nos permite, entre otras cosas, proseguir con el flujo de diseño, pero primero vamos a analizar los resultados de la síntesis. Podemos obtener información:



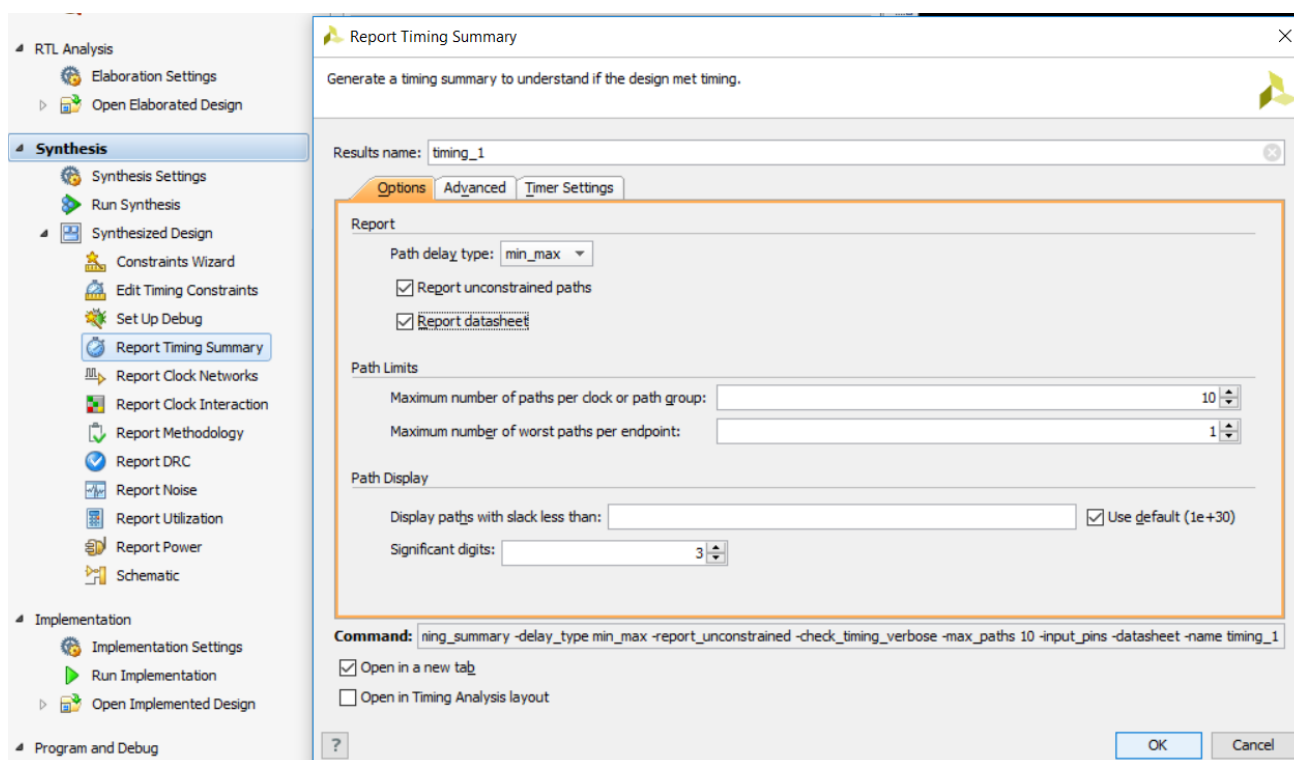
- En la pestaña **Project Summary** del área de visualización:  
Nos informa que se ha completado con éxito, sin errores ni warnings. Se dispone ya de una estimación de los recursos del FPGA que utilizará. (1 en la figura anterior)
- En la pestaña **Design Runs** de la ventana inferior:  
Nos informa de que el diseño ya está sintetizado aunque no implementado. También nos indica el tiempo que ha tardado la síntesis y los recursos del FPGA que utiliza. (2)
- En la pestaña **Reports** de la ventana inferior. Toda la información del proceso de síntesis se encuentra en el informe de síntesis.
- Abriendo el diseño sintetizado en el área **Flow Navigator (Open Synthesized Design)**. Esto nos permite una análisis más detallado de distintos aspectos e imponer/modificar las restricciones del diseño (1 en la figura siguiente) . Por ejemplo:
  - o Podemos “ver” el circuito sintetizado (picando dos veces en **schematic**)(2)
  - o Podemos hacer un análisis temporal. El entorno Vivado dispone de una herramienta de análisis temporal estático para la validación temporal de los diseños. En particular, si se han impuesto restricciones temporales al diseño, nos informa de su cumplimiento o violación (esto lo veremos en la Sesión 2).





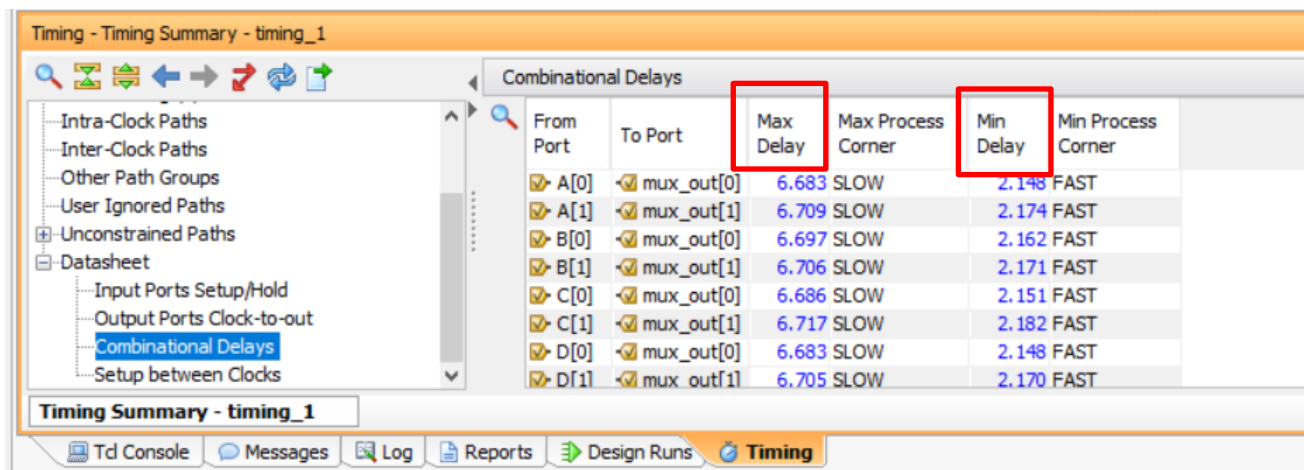
En esta sesión vamos a generar un informe sobre comportamiento temporal.

Seleccionamos **Synthesized Design -> Timing Report Summary**. Habilitamos la opción **Report Datasheet** y dejamos las restantes opciones por defecto:



La herramienta realiza un análisis temporal evaluando el retraso de los distintos caminos del circuito. La siguiente captura muestra los resultados de este análisis. Al tratarse de un circuito combinacional nos muestra el retraso mínimo y máximo desde cada entrada a cada salida (si existen realmente esos caminos de propagación de señal).





Timing - Timing Summary - timing\_1

Combinational Delays

From Port	To Port	Max Delay	Max Process Corner	Min Delay	Min Process Corner
A[0]	mux_out[0]	6.683	SLOW	2.148	FAST
A[1]	mux_out[1]	6.709	SLOW	2.174	FAST
B[0]	mux_out[0]	6.697	SLOW	2.162	FAST
B[1]	mux_out[1]	6.706	SLOW	2.171	FAST
C[0]	mux_out[0]	6.686	SLOW	2.151	FAST
C[1]	mux_out[1]	6.717	SLOW	2.182	FAST
D[0]	mux_out[0]	6.683	SLOW	2.148	FAST
D[1]	mux_out[1]	6.705	SLOW	2.170	FAST

Timing Summary - timing\_1

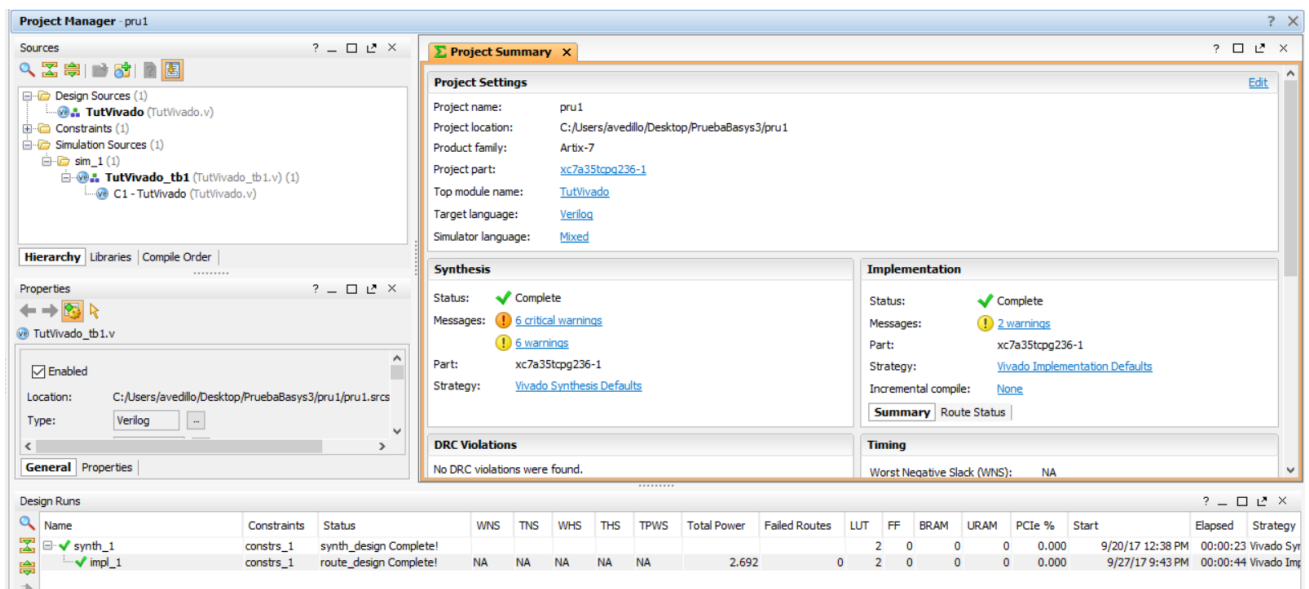
Td Console Messages Log Reports Design Runs Timing

### 3.6. Implementación

En la fase de implementación la lógica se distribuye en CLBs. Cada uno de ellos se coloca (*place*) en una determinada posición de la matriz de CLBs del FPGA objetivo y se conectan (*route*) usando los recursos de interconexión programables disponibles.

Seleccionamos el diseño y pulsamos dos veces en **Run Implementation** en el área del **Flow Navigator**. Pulsamos **OK** en la ventana que se abre. Una vez completada se abre una ventana que nos permite abrir el diseño implementado, proseguir en el flujo de diseño (**Generate Bitstream**) o analizar los informes generados (**View Reports**). Vamos a cancelar.

Como tras la síntesis, podemos obtener información de la implementación y del diseño a través del Project Summary o de los Reports.



Project Manager - pr1

Project Summary

Project Settings

Project name: pr1  
Project location: C:/Users/avedillo/Desktop/PruebaBasys3/pr1  
Product family: Artix-7  
Project part: xc7a35tqpg236-1  
Top module name: TutVivado  
Target language: Verilog  
Simulator language: Mixed

Synthesis

Status: Complete  
Messages: 6 critical warnings, 6 warnings  
Part: xc7a35tqpg236-1  
Strategy: Vivado Synthesis Defaults

Implementation

Status: Complete  
Messages: 2 warnings  
Part: xc7a35tqpg236-1  
Strategy: Vivado Implementation Defaults  
Incremental compile: None

DRC Violations

No DRC violations were found.

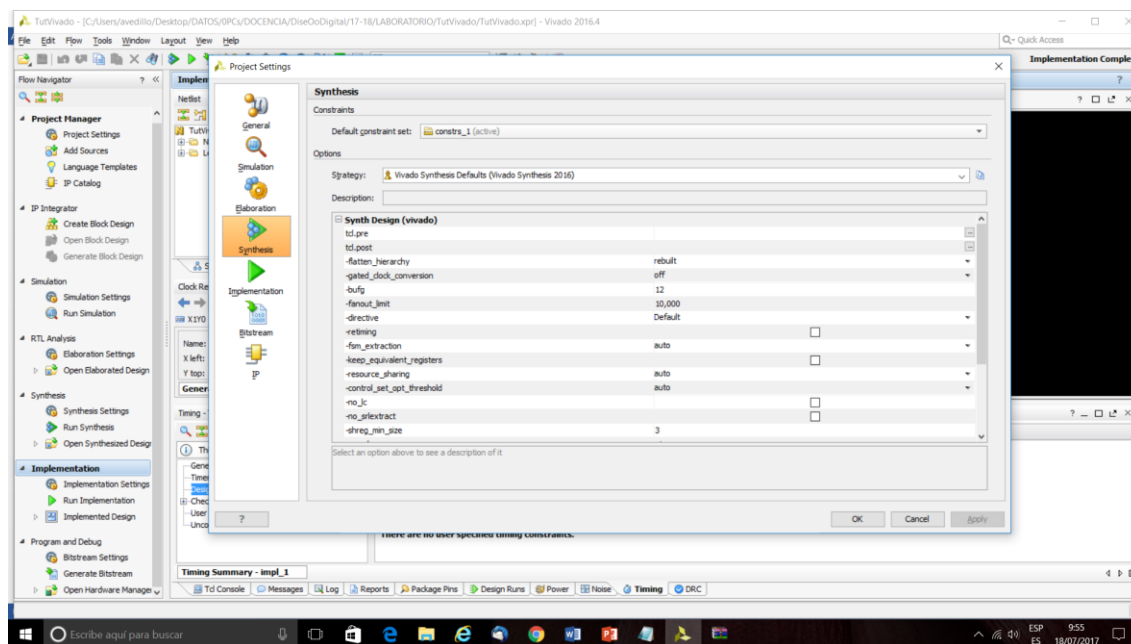
Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCIE %	Start	Elapsed	Strategy
synth_1	constrs_1	synth_design Complete!	NA	NA	NA	NA	NA	2.692	0	2	0	0	0	0.000	9/20/17 12:38 PM	00:00:23	Vivado Syn
impl_1	constrs_1	route_design Complete!	NA	NA	NA	NA	NA	2.692	0	2	0	0	0	0.000	9/27/17 9:43 PM	00:00:44	Vivado Im

Observamos que hay ahora más información en el **Project Summary**. En la pestaña **Design Runs** se informa de que el diseño ha sido sintetizado e implementado con éxito (marca verde a la izquierda). Abriendo el diseño implementado se habilitan capacidades de análisis más detallado.

### Más sobre síntesis e implementación:

- Los procesos de síntesis e implementación se pueden controlar configurando un conjunto de opciones.
- Una estrategia de síntesis (implementación) es una determinada configuración de las opciones de síntesis (implementación).
- Se accede a estas opciones en **Synthesis Settings (Implementation Settings)** en la ventana del flujo de diseño.



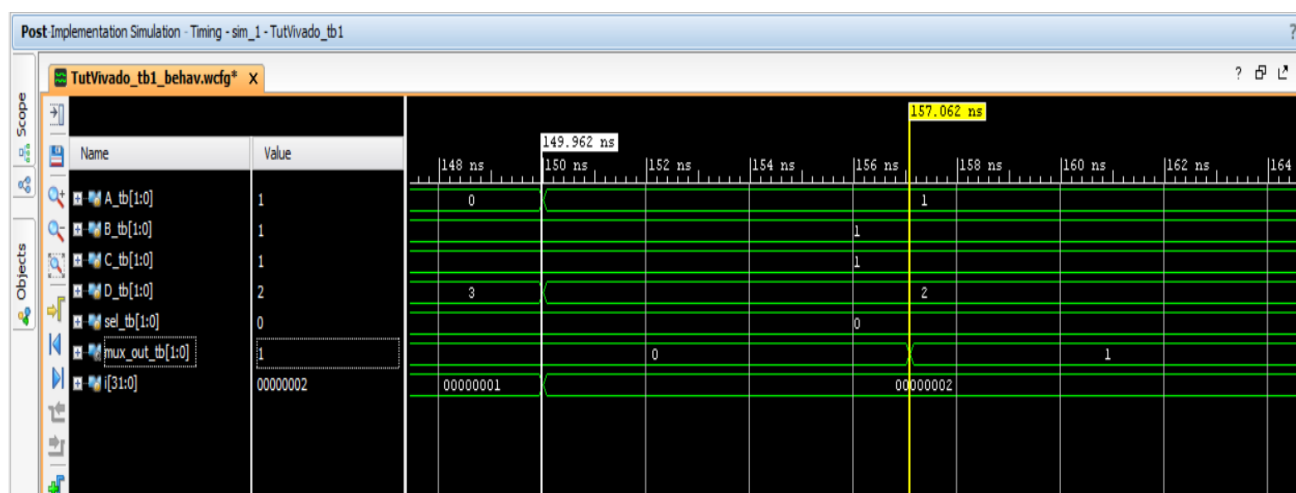
- Hay distintas estrategias predefinidas, pero el usuario también puede crear estrategias propias configurando las distintas opciones de síntesis o implementación.
- Es usual definir varios **Runs** asociando distintas restricciones, dispositivos físicos, o estrategias de síntesis e implementación como mecanismo de exploración del espacio de soluciones:
  - La ventana **Design Runs** muestra todos los runs creados y proporciona comandos para configurarlos, manejarlos y ejecutarlos. En particular informan de su estado: *not started*, *in progress*, *complete*, *out-of-date*.
  - Para crearlos: en la pestaña **Design Runs** de la consola, pulsamos **botón derecho -> Create Runs**
  - Un run de síntesis puede tener varios runs de implementación
  - Únicamente un run de síntesis y un run de implementación puede estar activo. Todos los report y ventanas de visualización muestran la información del run activo
  - Para hacer un run el activo: seleccionarlo, botón **Make Active**.

### 3..7. Simulación Post Implementation

Una vez completada la síntesis y la implementación, volvemos a realizar simulaciones para validar el diseño. Es posible hacer dos tipos de simulaciones en este punto del flujo de diseño:

**Funcional.** El objetivo es validar la funcionalidad del circuito que se va a implementar en el FPGA. Es decir, el modelo simulado no es ahora el código HDL inicial, sino un modelo generado tras haber sintetizado e implementado (netlist de recursos del FPGA). En el **Flow Navigator** seleccionamos **Run Simulation -> Run Post Implementation-> Functional Simulation**.

**Temporal (Timing).** El circuito que se va a implementar en el FPGA se simula incorporando información del comportamiento temporal de sus componentes y conexiones y que por lo tanto puede utilizarse para una validación temporal. En el **Flow Navigator** seleccionamos **Run Simulation -> Run Post Implementation -> Timing Simulation**.



Las formas de onda muestran ahora información sobre el comportamiento temporal. Los cambios en las salidas no son ahora simultáneos con los cambios de entrada. Observar el retraso desde el cambio de la entrada A de 0 a 1 (primer cursor) que provoca el cambio de la salida mux\_out (segundo cursor).

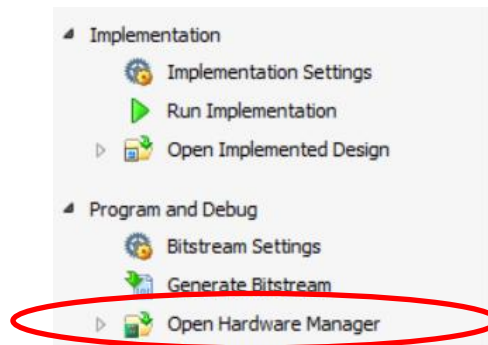
### 3..8. Generación del fichero de programación

Una vez completada la validación del diseño estamos en condiciones de generar el fichero de programación. Para ello, seleccionamos **Generate Bitstream** en el área del **Flow Navigator**. En la carpeta del proyecto aparece un fichero con extensión .bit que es el *bitstream* necesario para programar (configurar) el FPGA.

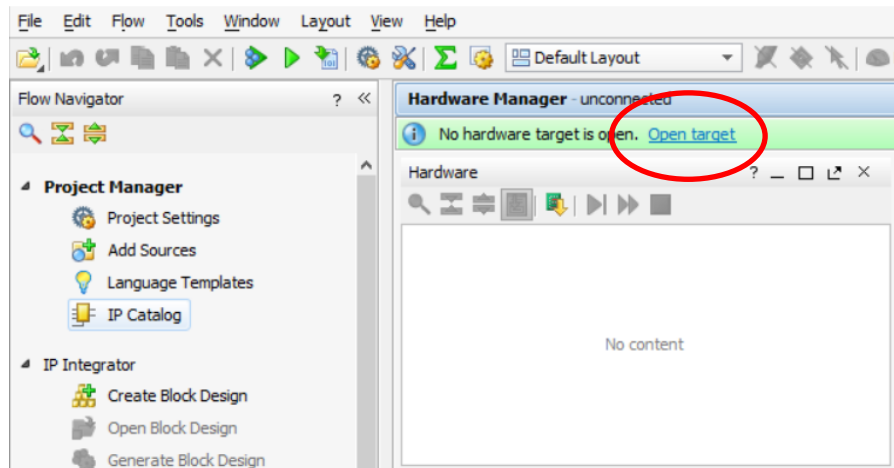
### 3..9. Programación del FPGA

Los pasos a seguir para programar el FPGA son:

1. Conectamos la placa al ordenador mediante el puerto USB.
2. Encendemos la placa si es necesario.
3. Pulsamos **Open Hardware Manager** en el apartado **Program and Debug** del **Flow Navigator**.

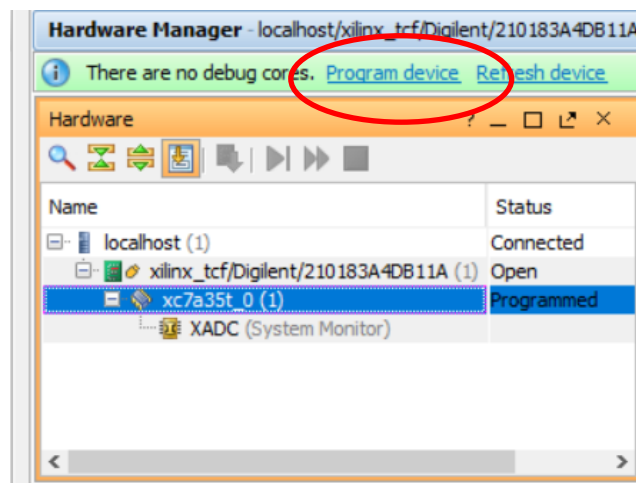


4. Pulsamos **Open target**:

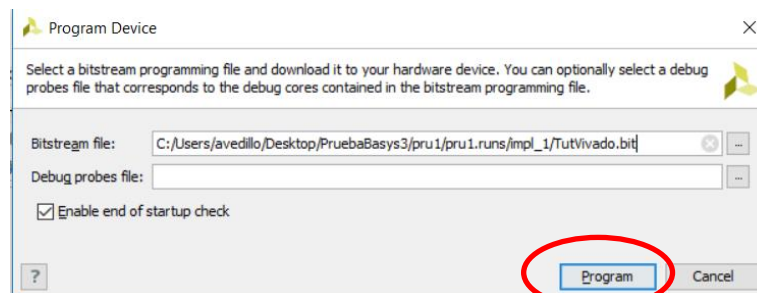


Seleccionamos **Auto Connect** en el desplegable que se abre.

5. Seleccionamos el FPGA y pulsamos **Program Device**:



6. Indicamos la ruta del archivo de programación (archivo .bit) y pulsamos **Program**:



## 4. Ejercicios

**S1.1.** Modifique el testbench del tutorial para realizar una simulación comportamiento en la que se aplique el conjunto completo de combinaciones de entrada. Se aplica una combinación distinta cada 25ns.

**S1.2.** Realice una simulación para determinar si el circuito secuencial descrito en Ejercicio\_S1\_2.v se comporta como un contador binario ascendente con *reset* síncrono activo en alto.

1. Añada la descripción al proyecto.
2. Cree un *testbench*. El *testbench* debe:
  - Describir un reloj de 50MHz
  - Comprobar que cuenta de acuerdo con la secuencia binaria natural ascendente.
  - Comprobar que se pone a cero si la señal de reset se pone a 1 y que lo hace de forma síncrona.
3. Simule (*Behavioural Simulation*) y analice los resultados de la simulación.

**S1.3.** Modifique la descripción anterior para que el contador sea disparado por flanco descendente, y tenga un *reset* asíncrono y activo en bajo. Realice una simulación para validar su comportamiento.

**S1.4.** Diseñe y simule un circuito combinacional con 4 entradas de 2 bits, A, B, C y D y una salida *Sal*, de 6 bits que implemente el siguiente comportamiento:

**Sal** = ((A \*B) + (C)) \*D;