

Práctica 6



Dispositivos Programables

Objetivo

Diseñar circuitos combinacionales en dispositivos programables FPGA.

Equipo

Computadora con el compilador Xilinx Vivado u otro software para desarrollo de código para FPGAs.

Fundamento teórico

Field Programmable Gate Array (FPGA)

Los dispositivos programables FPGA ofrecen un número grande de bloques lógicos (Logic blocks) que contienen circuitos de lógica combinacional y registrada que se pueden programar de forma independiente. También, contienen un conjunto de bloques de entrada y salida (I/O) que pueden ser configurados como entrada fija, salida fija o bidireccionales. Las salidas tienen capacidad para operar como de tres estados y los registros pueden ser usados para retener datos de entrada o salida. Una arquitectura general de FPGAs es mostrada en la Fig. 1. Todos los bloques lógicos y de entrada y salida pueden ser interconectados por programación para implementar virtualmente cualquier circuito. La capacidad de programar las interconexiones se logra por medio de líneas que circulan a través de los renglones y columnas en los canales entre los bloques lógicos.

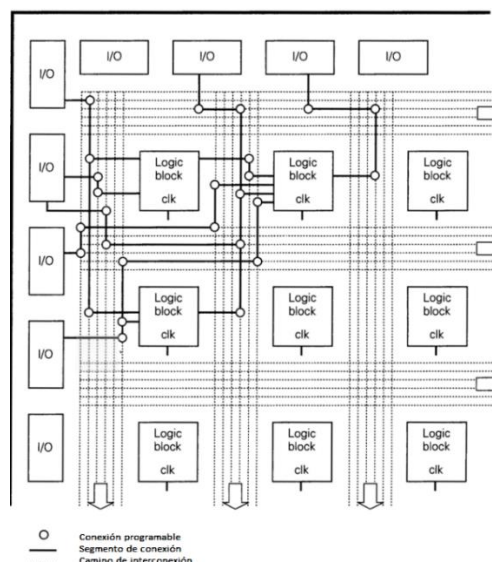


Figura 1. Arquitectura general de FPGAs.

Existen diferentes arquitecturas específicas de FPGAs y los datos que definen las conexiones programables son almacenados usando diferentes tecnologías, tales como SRAM, EEPROM, EEPROM flash y antifuse. El método de SRAM carga los datos a la RAM volátil desde una ROM externa cuando se conecta energía al dispositivo. Los métodos de EEPROM y flash trabajan de forma similar a los GAL. Tecnologías antifuse usan una conexión que está abierta hasta que se aplica un pulso de programación que causa que se pongan en corto. Es justo lo opuesto a fuse y de igual manera es reversible. Algunas FPGAs incluyen grandes bloques de memoria, otros no. Algunas usan arreglos de términos de producto para general expresiones SOP como los GAL, otros usan un enfoque de tabla de búsqueda (lookup tables, LUT). Como se puede ver, el campo de las FPGAs es diverso y cambia constantemente.

Arquitectura de FPGAs Xilinx

La estructura básica de una FPGA está compuesta de los siguientes elementos:

- **Look-up table (LUT):** Tabla de búsqueda, este elemento se encarga de las operaciones lógicas.
- **Flip-flop:** Este elemento de registro almacena los resultados del LUT.
- **Conexiones:** Conectan elementos entre sí.
- **Pads de Entrada/Salida (I/O):** Puertos físicos que envían datos dentro y fuera de la FPGA.

La combinación de estos elementos resulta en la arquitectura básica de FPGAs mostrada en la Fig. 2. A pesar de que esta estructura es suficiente para implementar cualquier algoritmo, la eficiencia de la implementación está limitada en términos de rendimiento computacional, recursos requeridos y frecuencia de reloj alcanzable.

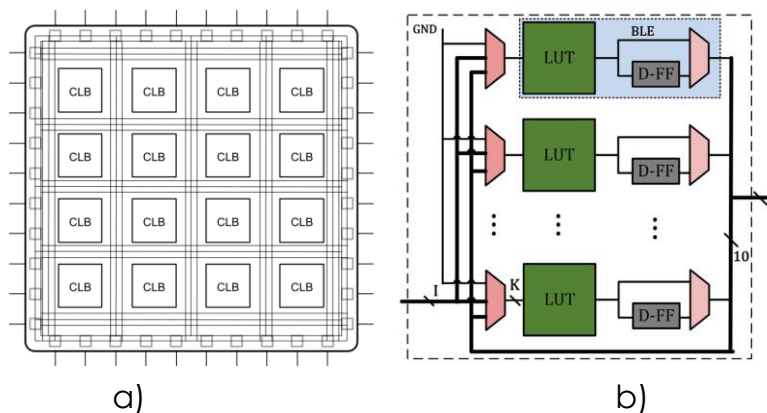


Figura 2. a) Estructura de una FPGA, b) Algunos componentes en un Configurable Logic Block (CLB).

FPGAs contemporáneas incorporan los elementos básicos junto con bloques adicionales de computación y almacenamiento que incrementan la densidad computacional y la eficiencia del dispositivo. Estos elementos adicionales son:

- Memorias embebidas para almacenamiento distribuido.
- Phase-locked loops (PLLs) para configurar la FPGA a diferentes velocidades de reloj.
- Transceivers serie de alta velocidad.
- Controladores de memoria.
- Bloques de Multiplicación-Suma.

La combinación de estos elementos provee a la FPGA con la flexibilidad de implementar cualquier algoritmo de software que se ejecuta en procesadores y resulta en la arquitectura contemporánea de FPGAs que se muestra en la Fig. 3.

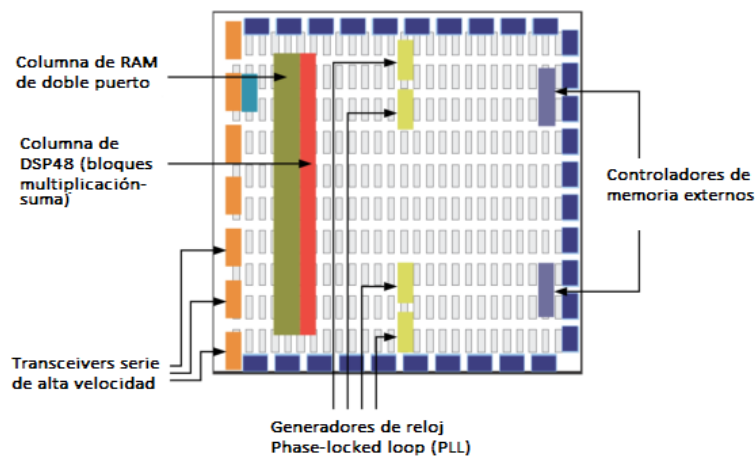


Figura 3. Arquitectura contemporánea de FPGAs.

LUT

Es el bloque de construcción básico en FPGAs y es capaz de implementar cualquier función lógica de N variables booleanas. Esencialmente, este elemento es una tabla de verdad en la cual diferentes combinaciones de las entradas implementan diferentes funciones para producir valores de salida. El límite en la tabla de verdad es N , donde N representa el número de entradas al LUT. Para el LUT general de N entradas, el número de localidades de memoria accedidas por la tabla es 2^N , el cual permite que la tabla implemente 2^{2^N} funciones. Un valor típico de N para FPGAs del fabricante Xilinx es 6.

La implementación en hardware de un LUT puede ser pensada como una colección de celdas de memoria conectadas a un conjunto de multiplexores. Las entradas al LUT actúan

como bits selectores en el multiplexor para seleccionar el resultado. Es importante tener esta representación en mente ya que un LUT puede ser usado tanto como un elemento de almacenamiento o como un motor de cálculo. La Fig. 4 muestra esta representación funcional del LUT.

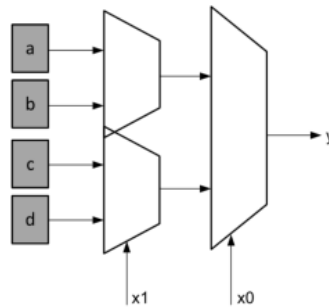


Figura 4. Representación funcional de un LUT como una colección de celdas de memoria.

Flip-flop

Es la unidad básica de almacenamiento dentro de una FPGA. Este elemento siempre está emparejado con un LUT para asistir en la canalización lógica y almacenamiento de datos.

Bloque DSP

Es el bloque computacional más complejo en una FPGA, el cual se muestra en la Fig. 5. Consiste en una Unidad Aritmética y Lógica (ALU) compuesta de una cadena de tres bloques diferentes: una unidad de suma/resta, conectada a un multiplicador, que está conectado a un motor de suma/resta/acumulación final. Esta cadena permite a una sola unidad DSP implementar funciones de la forma:

$$p = a \times (b + d) + c$$

ó

$$p += a \times (b + d)$$

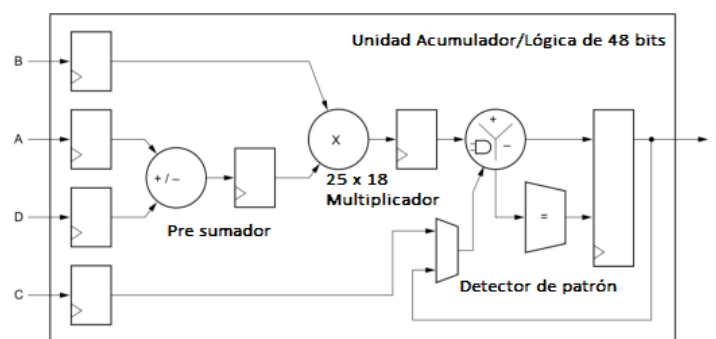


Figura 5. Estructura de un bloque DSP.

Elementos de almacenamiento

El dispositivo FPGA incluye elementos de memoria empotrados que pueden ser usados como memoria de acceso aleatorio (RAM), memoria de solo lectura (ROM) o registros de corrimiento. Estos elementos son bloques de RAM (BRAMs), bloques UltraRAM (URAMs), LUTs y registros de corrimiento (SRLs).

BRAMs pueden implementar ya sea una ROM o una RAM, la única diferencia es cuándo son escritos los datos. En una implementación de RAM, los datos pueden ser escritos y leídos en cualquier tiempo en la ejecución. En cambio, en una implementación de ROM, los datos solo se pueden leer durante la ejecución. Los datos se escriben en ROM como parte de la configuración del dispositivo y no pueden ser modificados.

Los bloques UltraRAM son RAM síncronas que proveen mucha más capacidad que las BRAM.

Como se describió anteriormente, el LUT es una pequeña memoria en la cual el contenido de una tabla de verdad es escrito durante la configuración del dispositivo. Debido a su flexibilidad, pueden ser usados como bloques de memoria, comúnmente llamados memorias distribuidas. Es la memoria más rápida en el dispositivo FPGA ya que puede ser instanciada en cualquier parte de su estructura.

FPGA y Procesadores

Un procesador, sin importar su tipo, ejecuta un programa como una secuencia de instrucciones, es decir, ejecuta una instrucción tras otra. La latencia computacional de cada instrucción no es igual entre tipos de instrucciones. Por ejemplo, dependiendo en donde se encuentre un operando, las instrucciones toman un número diferente de ciclos en completarse. Si está en la caché del procesador, la instrucción se puede completar en unas decenas de ciclos de reloj. Si está en memoria RAM DDR, las instrucciones podrían requerir cientos o miles de ciclos de reloj. Y si estuviera en el disco duro, las instrucciones tardarían aún más.

FPGAs son estructuras de inherente procesamiento paralelo capaces de implementar cualquier función aritmética o lógica que pueda ejecutarse en un procesador. Una FPGA también difiere de un procesador en la arquitectura de memoria y en el costo de los accesos a memoria. Los compiladores para FPGAs son capaces de organizar las memorias como múltiples bancos de almacenamiento lo más cercanos posibles al punto en donde se

van a usar en el circuito. Esto resulta en un instantáneo ancho de banda de memoria, que excede por mucho al de un procesador.

Field Programmable Analog Array (FPAA)

Es un circuito integrado con una colección de bloques de construcción análogos interconectados, que logran una reconfigurabilidad similar a las FPGAs. Son la contraparte análoga de FPGAs. A diferencia de ellas, las FPAAs tienden a estar orientadas a tareas específicas en lugar de ser de propósito general, se usan en circuitos tales como diseño de filtros.

Desarrollo

1. Cree una cuenta en www.xilinx.com.
2. Descargue el software de diseño con lenguajes de descripción de hardware **Xilinx Vivado Design Suite HL** versión 2020.1 de:
<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools.html>
Elija la versión de acuerdo al sistema operativo donde vaya a ejecutarlo.

Vivado Design Suite - HLx Editions - 2020.1 Full Product Installation

Important Information

Vivado Design Suite 2020.1 is now available:

- Ability to select the full image or selected products as part of Web installer
- Address map enhancements provide Realtime error highlighting and cross probing
- Nested DFX further extends the flexibility of DFX solutions
- Report QoR Suggestions predicts up to 3 custom strategies for Improved performance
- Power Rail based reporting now available

We **strongly recommend** to use the web installers as it reduces download time and saves significant disk space.

Please see [Installer Information](#) for details.

Note:

- Download verification is only supported with Google Chrome and Microsoft Internet Explorer web browsers.
- Beginning this release, the Single File Download and the Webinstaller supports all products. Vivado Lab Solutions and Document Navigator are included in both the Single File Download and Webinstaller packages.

Download Includes

Vivado Design Suite HLx Editions (All Editions)

Download Type

Full Product Installation

Last Updated

Jun 4, 2020

Answers

[2020.x - Vivado Known Issues](#)

Documentation


[Release Notes](#)
[OS Support Update](#)
[What's New in Vivado](#)

Support Forums

[Installation and Licensing](#)

 [Xilinx Unified Installer 2020.1: Windows Self Extracting Web Installer \(EXE - 66.73 MB\)](#)

MD5 SUM Value : e4339ae3bcad478d7130edd669aac786

Download Verification 

Digests

Signature

Public Key

 [Xilinx Unified Installer 2020.1: Linux Self Extracting Web Installer \(BIN - 116.89 MB\)](#)

MD5 SUM Value : 1f21c8a5858b947c003f741826b5bce5

Download Verification 


Digests

Signature

Public Key

 [Vivado HLx 2020.1: All OS installer Single-File Download \(TAR/GZIP - 35.51 GB\)](#)

MD5 SUM Value : b018f7b331ab0446137756156ff944d9

Download Verification 

Digests

Signature

Public Key

Para Windows

Para Linux

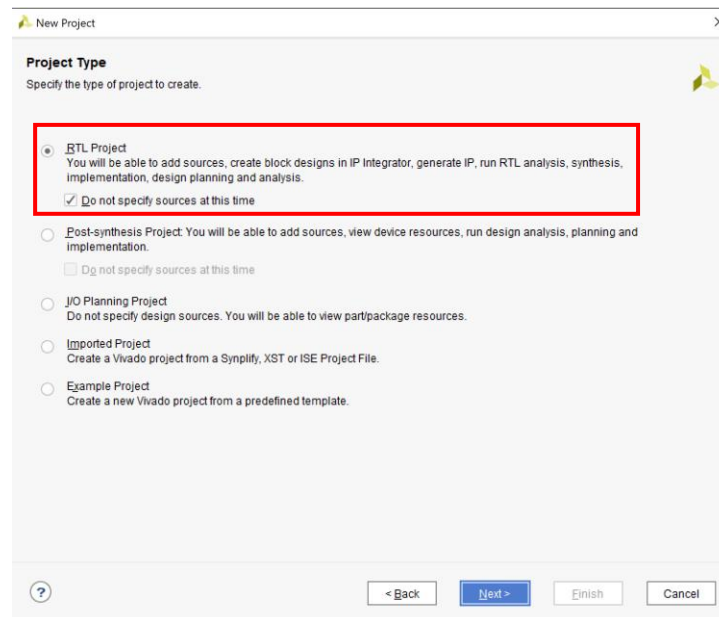
3. Instale el software. Seleccione la licencia gratuita **WebPACK**, la cual es suficiente para los programas que se van a desarrollar en clase, además de que incluye muchas de las opciones disponibles en las licencias comerciales.

Download Vivado HL WebPACK Edition

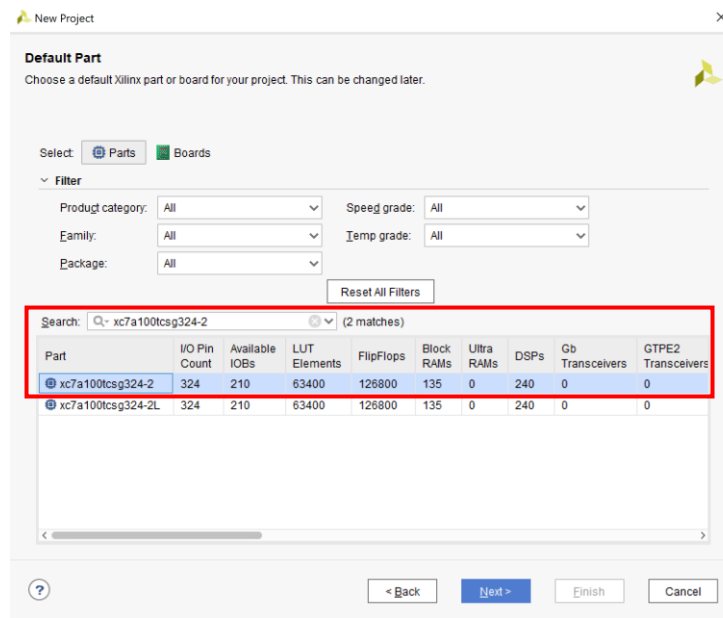
The Vivado Design Suite HL WebPACK™ Edition is the FREE version of the revolutionary design suite. Vivado HL WebPACK delivers instant access to some basic Vivado features and functionality at no cost.

Vivado Design Suite - HLx Edition Features	Vivado HL Design Edition	Vivado HL System Edition	Vivado Lab Edition	Vivado HL WebPACK (Device Limited)	Free 30-day Evaluation
Accelerating Implementation					
Synthesis and Place and Route	✓	✓		✓	✓
Dynamic Function Exchange	✓	✓		✓	✓
Accelerating Verification					
Vivado Simulator	✓	✓		✓	✓
Vivado Device Programmer	✓	✓	✓	✓	✓
Vivado Logic Analyzer	✓	✓	✓	✓	✓
Vivado Serial I/O Analyzer	✓	✓	✓	✓	✓
Debug IP (ILA/VIO/IBERT)	✓	✓		✓	✓
Accelerating High Level Design					
Vivado High-Level Synthesis	✓	✓		✓	✓
Vivado IP Integrator	✓	✓		✓	✓
System Generator for DSP		✓			✓

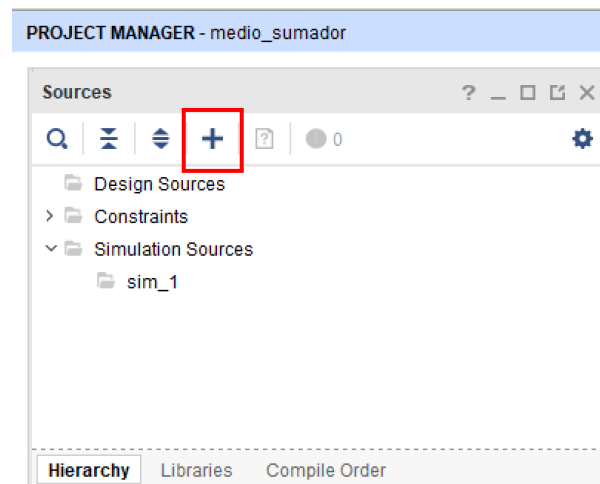
4. Ejecute **Vivado** y cree un nuevo proyecto llamado **medio_sumador**, seleccione el tipo de proyecto **RTL**.

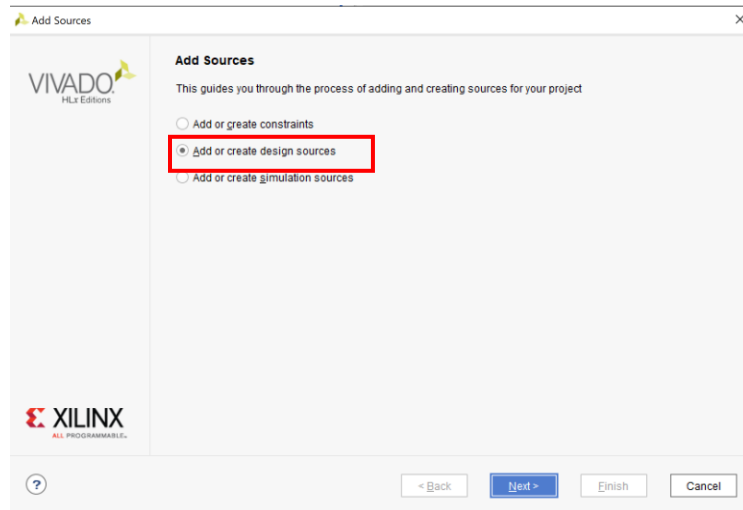


En el campo **tipo de parte** escriba y seleccione **xc7a100tcsg324-2**, una FPGA Xilinx de la familia Artix 7.

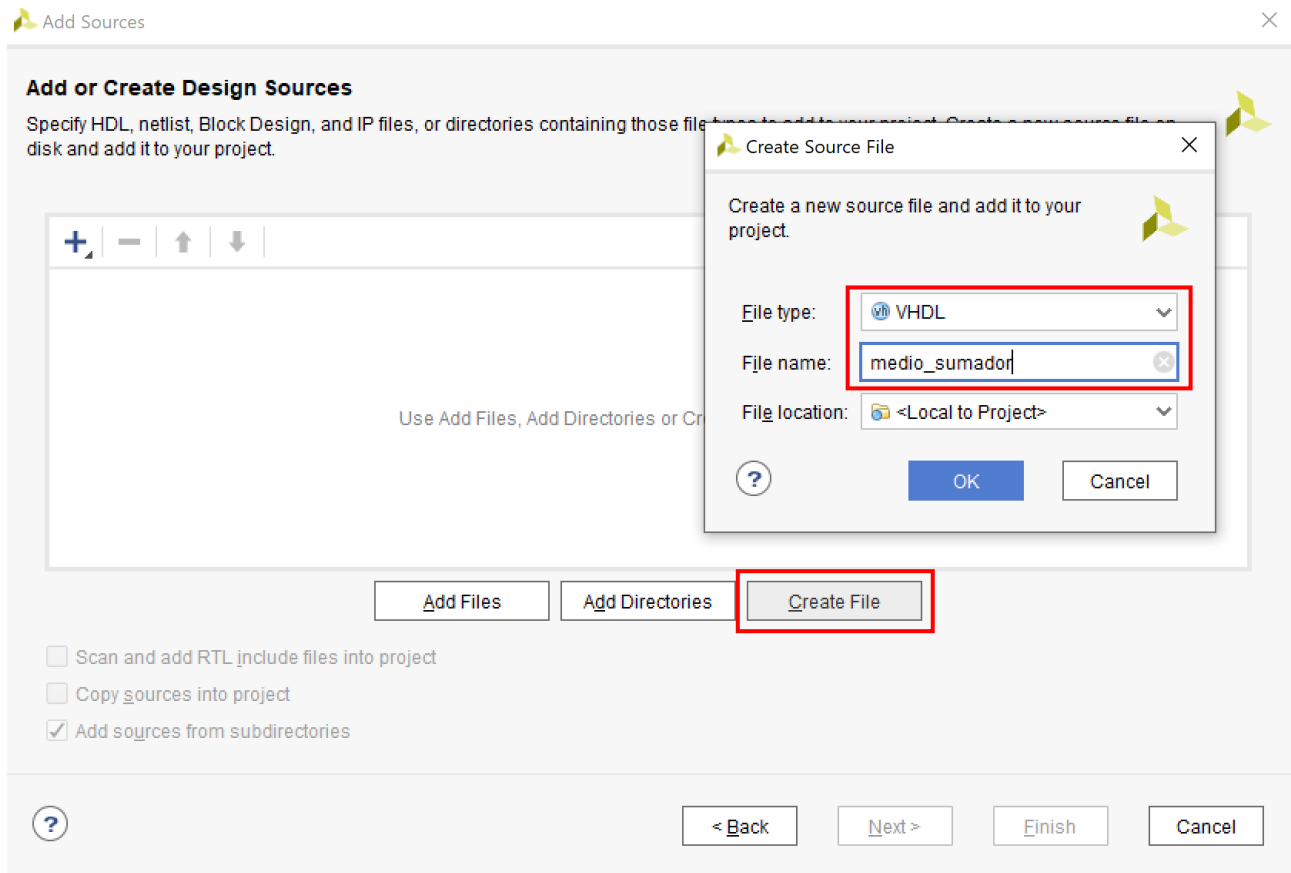


5. Cree un archivo de código fuente **VHDL**. En el área de Project Manager, haga clic en **Add Sources** y seleccione **Add or create design sources**.

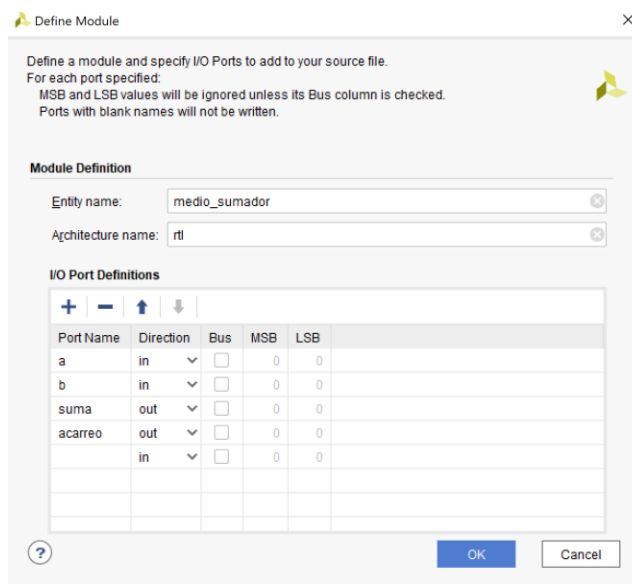




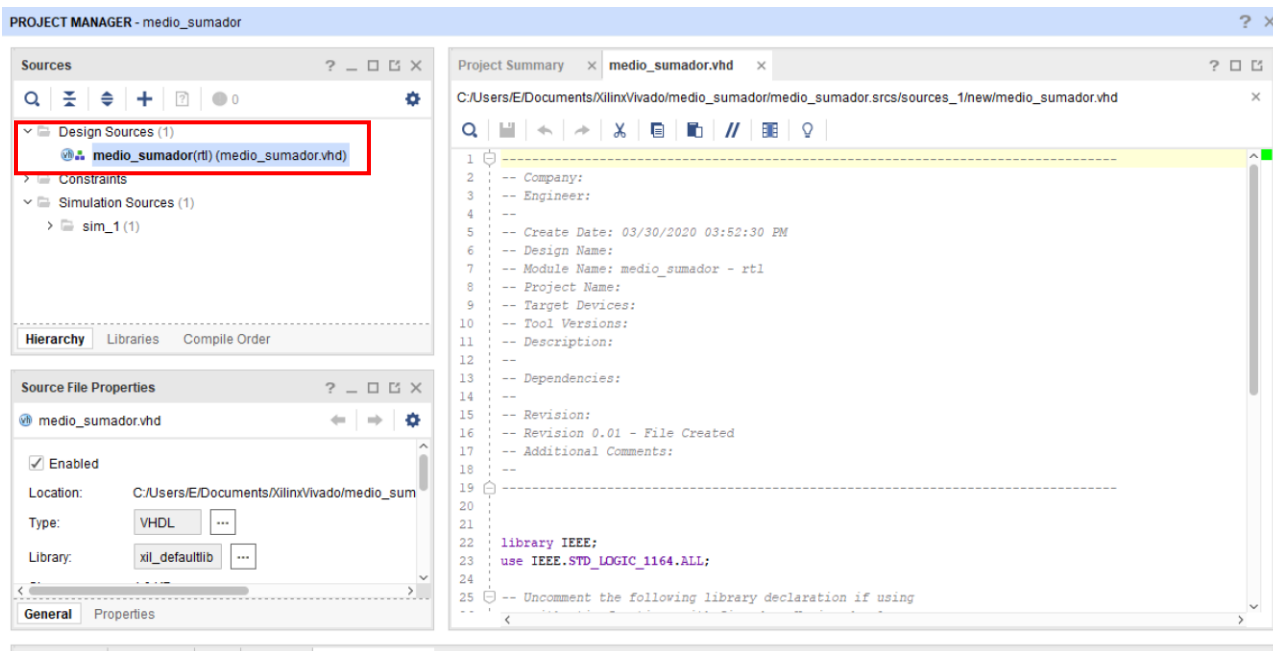
6. Cree un nuevo archivo llamado **medio_sumador** de código **VHDL**.



7. Agregue dos entradas y dos salidas al diseño, como se ve en la siguiente figura:



8. El archivo de código **VHDL** se va a agregar a la carpeta de diseño. Abra el archivo por medio de dar doble clic en **medio_sumador(rtl)(medio_sumador.vhdl)**.



9. Describa un medio sumador en **VHDL**. Para esto, agregue las siguientes dos líneas de código entre las sentencias **begin** y **end**:

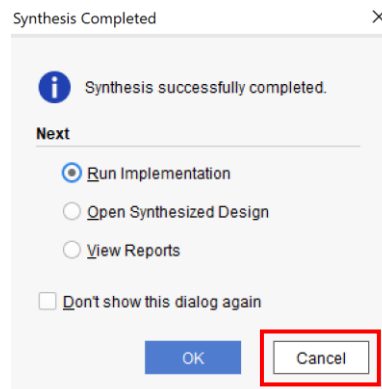
```
suma    <= a xor b;  
acarreo <= a and b;
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
  
-- Uncomment the following library declaration if using  
-- arithmetic functions with Signed or Unsigned values  
--use IEEE.NUMERIC_STD.ALL;  
  
-- Uncomment the following library declaration if instantiating  
-- any Xilinx leaf cells in this code.  
--library UNISIM;  
--use UNISIM.VComponents.all;  
  
entity medio_sumador is  
    Port ( a : in STD_LOGIC;  
          b : in STD_LOGIC;  
          suma : out STD_LOGIC;  
          acarreo : out STD_LOGIC);  
end medio_sumador;  
  
architecture rtl of medio_sumador is  
  
begin  
    suma    <= a xor b;  
    acarreo <= a and b;  
  
end rtl;
```

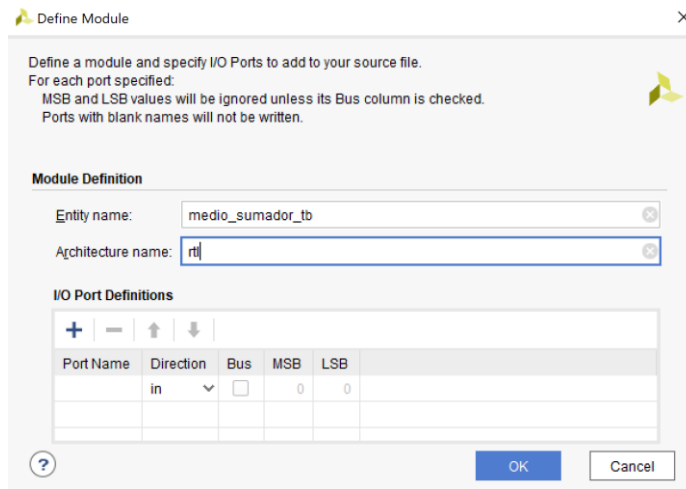
10. Sintetice el código. **Sintetizar** consiste en convertir un código escrito en un lenguaje de descripción de hardware (Hardware Description Language, HDL), tal como VHDL o Verilog, a una lista de conexiones (netlist) en términos de compuertas lógicas. Para sintetizar el código del medio sumador, seleccione **Run Synthesis**.



El diseño solo se va a simular, por lo que no es necesario ejecutar la implementación. En la ventana que indica que la síntesis ha terminado con éxito, haga clic en **Cancel**.



11. Cree un archivo de pruebas (test bench) para la simulación. Nuevamente, haga clic en **Add Sources** y seleccione **Add or create design sources**. Cree un archivo llamado **medio_sumador_tb** de código **VHDL** y no agregue puertos de entrada y salida.



Define a module and specify I/O Ports to add to your source file.
For each port specified:
MSB and LSB values will be ignored unless its Bus column is checked.
Ports with blank names will not be written.

Module Definition

Entity name: medio_sumador_tb

Architecture name: rtl

I/O Port Definitions

Port Name	Direction	Bus	MSB	LSB
	in	<input type="checkbox"/>	0	0

OK Cancel

12. Agregue al test bench el componente del medio sumador que se describió en **medio_sumador.vhdl**, cree señales y una instancia del medio sumador en donde enlace las señales.

Componente:

```
component medio_sumador
Port(
    a : in std_logic;
    b : in std_logic;
    suma : out std_logic;
    acarreo : out std_logic
);
end component;
```

Señales:

```
signal s_a : std_logic := '0';
signal s_b : std_logic := '0';
signal s_suma : std_logic;
signal s_acarreo : std_logic;
```

Instanciación:

```
uut : medio_sumador
port map (
    a => s_a,
```

```

b    => s_b,
suma    => s_suma,
acarreo => s_acarreo
);

```

Agregue al test bench un proceso donde se asignen a las entradas los distintos estados que pueden presentar y se mantenga cada estado por 100 ns.

```

process is
begin
    s_a <= '0';
    s_b <= '0';
    wait for 100 ns;
    s_a <= '0';
    s_b <= '1';
    wait for 100 ns;
    s_a <= '1';
    s_b <= '0';
    wait for 100 ns;
    s_a <= '1';
    s_b <= '1';
    wait for 100 ns;
end process;

```

Su test bench va a quedar estructurado de la siguiente manera:

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity medio_sumador_tb is
-- Port ( );
end medio_sumador_tb;

architecture rtl of medio_sumador_tb is

    component medio_sumador
    Port(
        a : in std_logic;
        b : in std_logic;
        suma : out std_logic;
        acarreo : out std_logic
    );
    end component;

    signal s_a : std_logic := '0';
    signal s_b : std_logic := '0';
    signal s_suma : std_logic;
    signal s_acarreo : std_logic;

begin

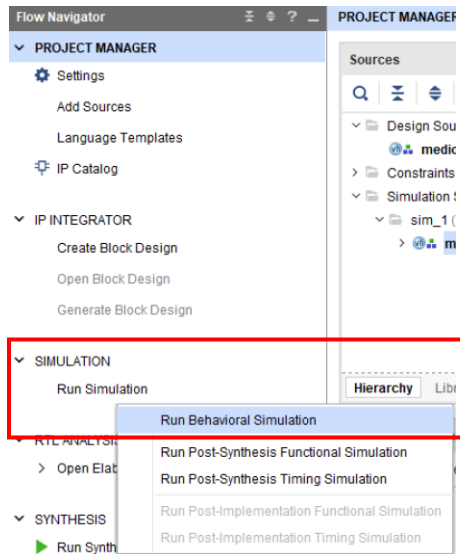
    uut : medio_sumador
    port map (
        a => s_a,
        b => s_b,
        suma => s_suma,
        acarreo => s_acarreo
    );

    process is
    begin
        s_a <= '0';
        s_b <= '0';
        wait for 100 ns;
        s_a <= '0';
        s_b <= '1';
        wait for 100 ns;
        s_a <= '1';
        s_b <= '0';
        wait for 100 ns;
        s_a <= '1';
        s_b <= '1';
        wait for 100 ns;
    end process;

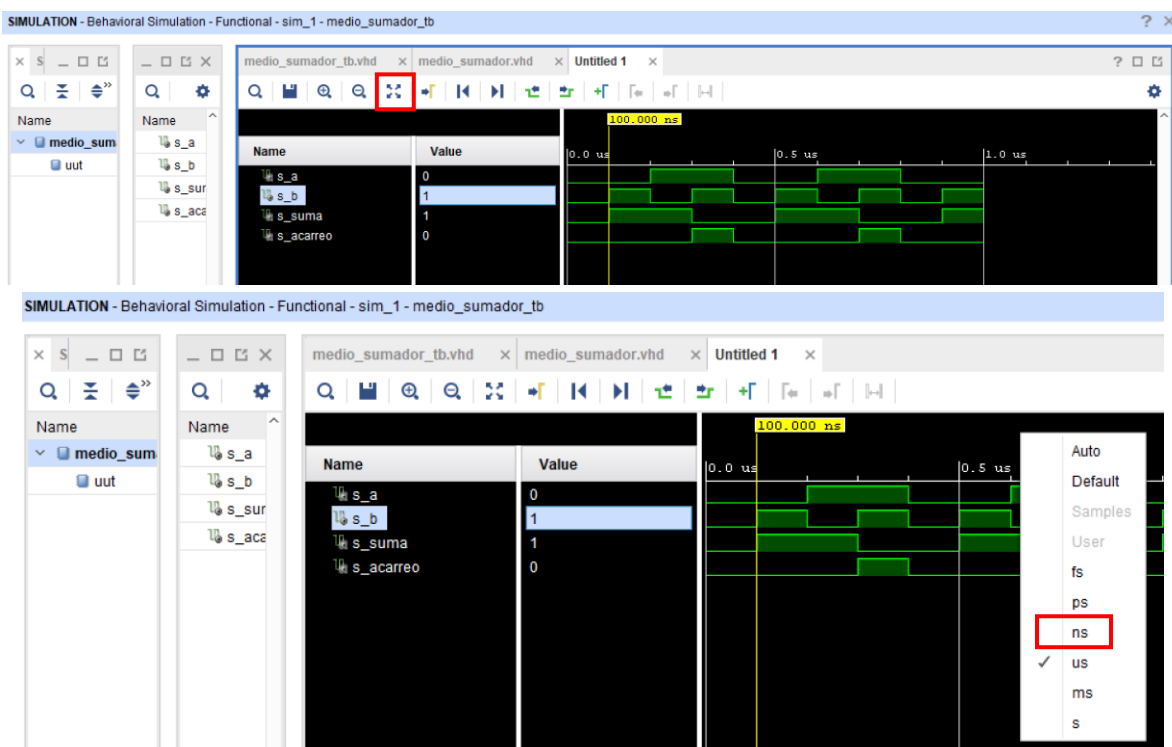
end rtl;

```


13. Simule el medio sumador. Haga clic en **Run Simulation** y seleccione **Run Behavioral Simulation**.



14. Se va a abrir una ventana con los resultados de la simulación, tal como se muestra en la siguiente figura. Para una mejor visualización de los estados de las entradas y salidas, haga clic en **Zoom Fit**. Si lo desea, también puede cambiar la resolución del diagrama de tiempos a **ns**, haciendo clic derecho en el diagrama.



15. Revise el video sobre una introducción a **VHDL** en:
<https://youtu.be/Olj59kyR7wU>
16. Cree un nuevo proyecto llamado **sumador_completo**, en el cual describa en **VHDL** un sumador completo con circuitería combinacional. Cree su test bench **sumador_completo_tb** y simule el código.
17. Realice un video donde describa el diagrama de tiempos de la simulación del sumador completo. Describa claramente en el diagrama los estados que toman las entradas y las salidas que producen. Indique con el cursor los tiempos en los que cambian de estado las entradas y salidas.
18. Entregue su reporte y proyecto de Vivado (u otra herramienta de desarrollo en VHDL para FPGAs) por Moodle, el video compártalo por medio de un enlace al drive de su cuenta universitaria.

Investigación:

- a) El proceso de diseño de circuitos en FPGAs Xilinx consiste en las etapas de **Síntesis** e **Implementación**, donde esta última se divide a su vez en: **Traducción** (Translate), **Mapeo** (Map) y **Ubicación y Ruteo de componentes** (Place and route). Describa detalladamente en qué consiste cada una de las etapas (las 5 etapas), qué reciben como entrada (código, netlist, etc.) y qué producen como salida (esquema a nivel de compuertas, mapeo a nivel de componentes específicos en el dispositivo como LUTs, BRAMS..., etc.).
- b) Realice un mapa conceptual donde indique las características y capacidades del dispositivo FPGA que se usó en esta práctica. Para las capacidades, puede basarse en el documento en: <https://www.xilinx.com/support/documentation/selection-guides/7-series-product-selection-guide.pdf>
El cual provee una comparativa de las FPGAs serie 7 de Xilinx.
- c) Finalmente, ¿qué es un archivo test bench y por qué es importante realizarlo?

Conclusiones y comentarios

Dificultades en el desarrollo

Referencias