

Manual de Prácticas para la tarjeta PYNQ – Z2

JULIO 2021

NOMBRE DE LA COMPAÑÍA

Creado por: Claudio Guadalupe Cruz Mendoza



Nombre del
logotipo

Contenido

MARCO TEÓRICO	4
PYNQ Z2	4
<i>Vivado</i>	4
<i>Vitis HLS</i>	5
<i>Vitis</i>	5
PYNQ	5
<i>Librerías</i>	5
<i>Asyncio</i>	6
<i>Time</i>	6
<i>Overlays</i>	6
CREACIÓN DE CUENTA EN XILINX	6
PREPARATIVOS PARA UTILIZAR LA TARJETA PYNQ Z2	15
INSTALACIÓN DE LA TARJETA	15
CREAR PROYECTO EN VIVADO	16
AÑADIR ARCHIVOS AL PROYECTO	19
SÍNTESIS – IMPLEMENTACIÓN -BITSTREAM	24
PRÁCTICA 1: ZYNQ CON MÓDULO RTL	25
OBJETIVOS	25
PROCEDIMIENTO	25
DESARROLLO	26
<i>Vivado</i>	26
<i>Vitis/SDK</i>	38
RESULTADO	42
PRÁCTICA 2: PRIMER BOOT	43
OBJETIVOS	43
DESARROLLO	43
<i>Configuración de la tarjeta</i>	44
RESULTADO	45
PRÁCTICA 3: CREACIÓN DE UN OVERLAY	45
OBJETIVO	45
DESARROLLO	46
<i>Diseño en Vivado</i>	46
<i>Uso de Jupyter Notebook</i>	47
<i>Guardar archivos en la tarjeta</i>	47
<i>Importar Overlay</i>	49
<i>Definir variables</i>	50
<i>Programar</i>	50
RESULTADO	51
PRÁCTICA 4: CREAR IP CON VITIS HLS	52
OBJETIVO	52
DESARROLLO	52
<i>Crear Proyecto</i>	52
<i>Realizar Código en C</i>	55

Ejecutar Síntesis y exportar IP	55
RESULTADO	56
PRÁCTICA 5: OVERLAY PERSONALIZADO CON IP'S.....	56
OBJETIVO	56
DESARROLLO.....	56
<i>Vivado</i>	57
<i>Jupyter</i>	61
<i>SDK</i>	64
RESULTADOS.....	66
<i>Python</i>	66
<i>SDK</i>	66
PRÁCTICA 6: OVERLAY CON MÚLTIPLES INTERRUPCIONES.....	67
OBJETIVO	67
DESARROLLO.....	67
<i>Jupyter</i>	68
RESULTADOS.....	71
PRÁCTICA 7: SUMA DE MATRICES	71
OBJETIVO	71
DESARROLLO.....	71
<i>Vitis HLS</i>	72
<i>Vivado</i>	73
<i>Jupyter</i>	75
RESULTADO	77
PRÁCTICA 8: MULTIPLICACIÓN DE MATRICES	78
OBJETIVO	78
DESARROLLO.....	78
<i>Vitis HLS</i>	78
<i>Vivado</i>	81
<i>SDK o Vitis</i>	81
RESULTADOS.....	83
PRACTICA 9: ZYNQ CON AXI TIMER	83
OBJETIVO	83
DESARROLLO.....	84
<i>Vivado</i>	84
<i>Vitis/SDK</i>	85
<i>Python</i>	89
RESULTADOS.....	92
<i>Python</i>	92
<i>SDK</i>	93
PRACTICA 10: ZYNQ CON HDMI	93
OBJETIVO	93
DESARROLLO.....	93
<i>Vivado</i>	94
RESULTADOS.....	95
ANEXO:CODIGO C - GPIO	95

Marco Teórico

Pynq Z2

La tarjeta TUL PYNQ-Z2, basada en el SoC Zynq de Xilinx, admite el framework PYNQ (Python Productivity for Zynq).

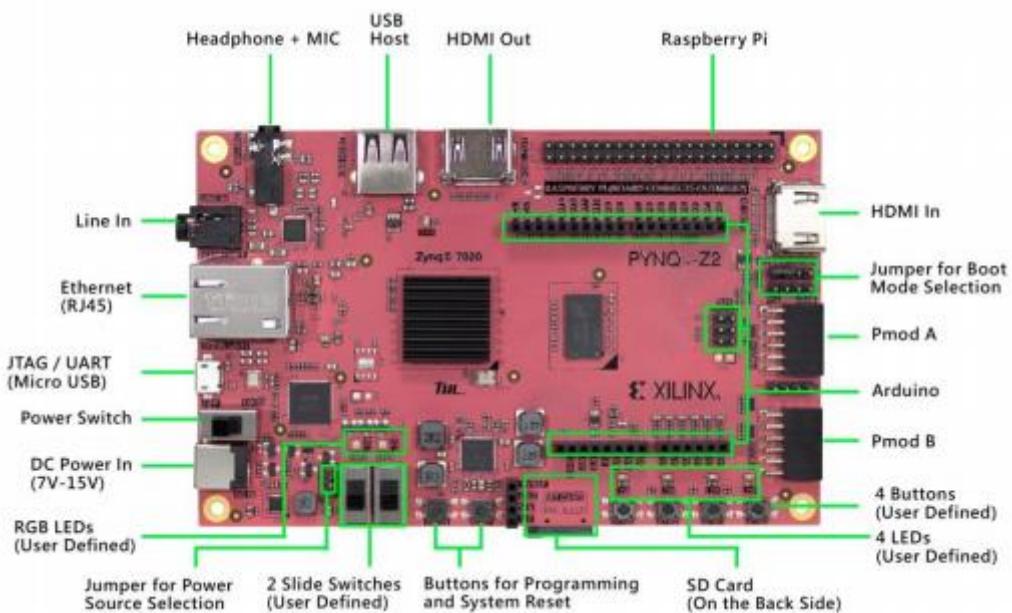


Figura 1 Tarjeta PYNQ Z2

Vivado Design Suite

Es un paquete de software producido por Xilinx para la síntesis y análisis de diseños HDL, que reemplaza a Xilinx ISE con características adicionales para el desarrollo de sistemas en un chip y síntesis de alto nivel.

Vivado

Es un componente de Vivado Design Suite. Es un simulador de lenguaje compilado que admite scripts Tcl en lenguaje mixto, IP encriptada y verificación mejorada.



Vitis HLS

El compilador Vitis High-Level Synthesis permite que los programas C, C ++ y SystemC se dirijan directamente a los dispositivos Xilinx sin la necesidad de crear un RTL manualmente. Vivado HLS ha sido ampliamente revisado para aumentar la productividad del desarrollador, y se confirma que admite clases, plantillas, funciones y sobrecarga de operadores de C ++. También tiene soporte para convertir automáticamente kernels OpenCL a IP para dispositivos Xilinx.



Vitis

PYNQ.

PYNQ es un proyecto de código abierto de Xilinx que facilita el uso de las plataformas Xilinx. Usando el lenguaje y las bibliotecas de Python, los diseñadores pueden aprovechar los beneficios de la lógica programable y los microprocesadores para construir sistemas electrónicos más capaces.

PYNQ se puede utilizar con Zynq, Zynq UltraScale +, Zynq RFSoC, placas de aceleración Alveo y AWS-F1 para crear aplicaciones de alto rendimiento con:

- Ejecución de hardware en paralelo
- Procesamiento de video de alta velocidad de cuadros.
- Algoritmos acelerados por hardware
- Procesamiento de señales en tiempo real
- Entradas / Salidas de gran ancho de banda
- Control de baja latencia

Librerías

Una biblioteca es una colección de códigos combinados previamente que se pueden utilizar de forma iterativa para reducir el tiempo necesario para codificar. Son particularmente útiles para acceder a los códigos preescritos de uso frecuente, en lugar de escribirlos desde cero cada vez. Al igual que las bibliotecas físicas, estas son una colección de recursos reutilizables, lo que significa que cada biblioteca tiene una fuente raíz.

Asyncio

Asyncio es una biblioteca para escribir código concurrente utilizando la sintaxis `async/await`. Es utilizado como base en múltiples frameworks asíncronos de Python y provee un alto rendimiento en redes y servidores web, bibliotecas de conexión de base de datos, colas de tareas distribuidas, etc. Asyncio suele encajar perfectamente para operaciones con límite de E/S y código de red estructurado de alto nivel.

Hello World!

```
import asyncio

async def main():
    print('Hello ...')
    await asyncio.sleep(1)
    print('... World!')

# Python 3.7+
asyncio.run(main())
```

Figura 2 Ejemplo de código utilizando la librería asyncio.

Time

Este módulo proporciona varias funciones relacionadas con el tiempo. Aunque este módulo siempre está disponible, no todas las funciones están disponibles en todas las plataformas. La mayoría de las funciones definidas en este módulo llaman a la plataforma de biblioteca C funciones con el mismo nombre.

Overlays

Los overlays, o bibliotecas de hardware, son diseños de FPGA programables configurables que amplían la aplicación del usuario desde el sistema de procesamiento del Zynq a la lógica programable. Los overlays se pueden utilizar para acelerar una aplicación de software o para personalizar la plataforma de hardware para una aplicación en particular.

Creación de cuenta en Xilinx

Para la descarga del software es necesario crear una cuenta en Xilinx, esta cuenta se puede crear con el correo institucional o bien con un correo personal. Para esto es necesario ingresar a la página <https://www.xilinx.com>.

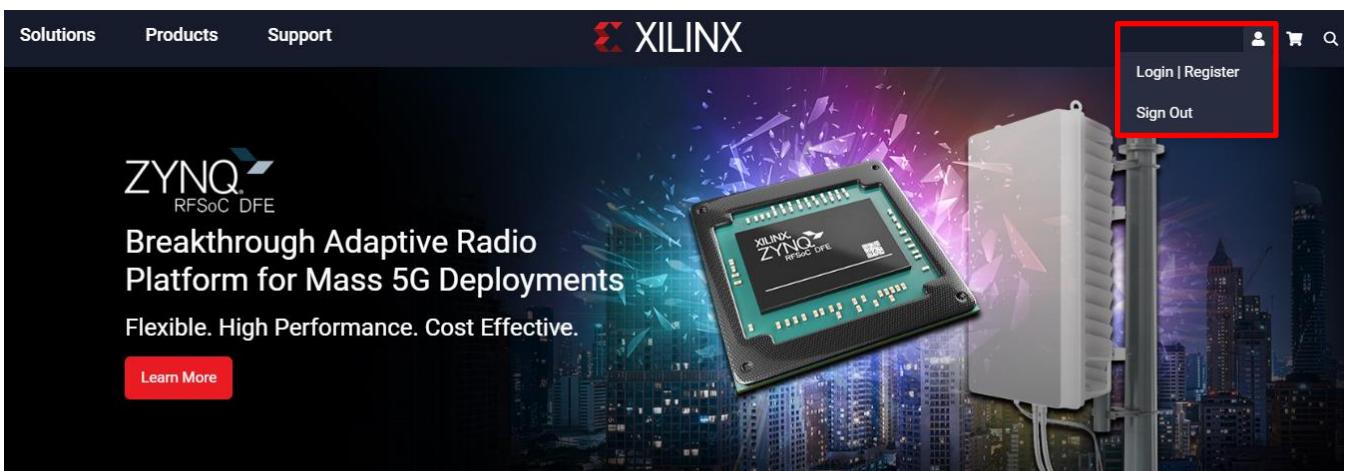


Figura 3 Pagina de Xilinx.

En la Figura 3, en el cuadro rojo, se encuentra la sección para iniciar sesión y registrar, dando clic en ella, cargara la página que se encuentra en la Figura 4.

A screenshot of the Xilinx 'Sign-In' page. The page has a header with the Xilinx logo. Below the header, there is a form with two input fields: 'E-mail Address' and 'Password'. Both fields have validation messages: 'Please enter an e-mail address' and 'Please enter a password'. Below the form is a large red 'Sign In' button. Underneath the button is a dark blue 'Create Account' button. At the bottom of the page, there are links for 'Forgot / Reset Password?' and 'Resend account activation e-mail?'. The entire page is contained within a light gray frame.

Figura 4 Pagina para iniciar sesión -crear cuenta.

Después de dar clic en "Create Account" se cargará la página que se muestra en la Figura 5, cabe mencionar que el registrarse con un correo personal podría limitar el acceso al soporte al cliente, evaluaciones de producto y sitios seguros.

Xilinx Account Creation

To create an account, complete and submit the fields below.

An activation e-mail confirmation will be automatically sent to your specified e-mail address.

First Name*

Last Name*

Corporate E-mail*

Note: Use of a non-company e-mail address may limit your access to customer support, product evaluations, and secure sites.

Location*

Figura 5 Formulario para crear una cuenta en Xilinx, el cuadro rojo resalta las limitaciones al usar un correo personal.

Una vez llenado el formulario anterior, se cargara la página que se aprecia en la Figura 6, en donde nos indica que se envió un correo al email registrado, en este se vendrá un token único de acceso con duración de 1 día, se ingresa y se establece una contraseña que cumpla con los requisitos(mínimo de 8 caracteres, debe contener al menos una letra mayúscula y minúscula, un número y un carácter especial) para la cuenta de Xilinx.

Next Step - activate your account

Please check your e-mail for a Xilinx account activation message.

To activate your account, enter the provided **Access Token** from your email, create and confirm a unique password, then proceed to Activate Account.

Access Token*

Password*

Confirm Password*

Password Strength: Must contain a minimum of 8 characters.

- Must contain a minimum of 8 characters and a maximum of 72 characters
- Must contain at least 1 lowercase letter, 1 uppercase letter, 1 number and 1 special character

Feedback

Note: If you do not receive a confirmation e-mail within a few minutes please check your junk mail folder and add sender noreply@xilinx.com to

Figura 6 Activación de la cuenta y creación de la contraseña.

Después de finalizar el proceso, se volverá a cargar la página de la Figura 4, donde ya con la cuenta activa se debe iniciar sesión, posteriormente se volverá a cargar la página de la Figura 3.

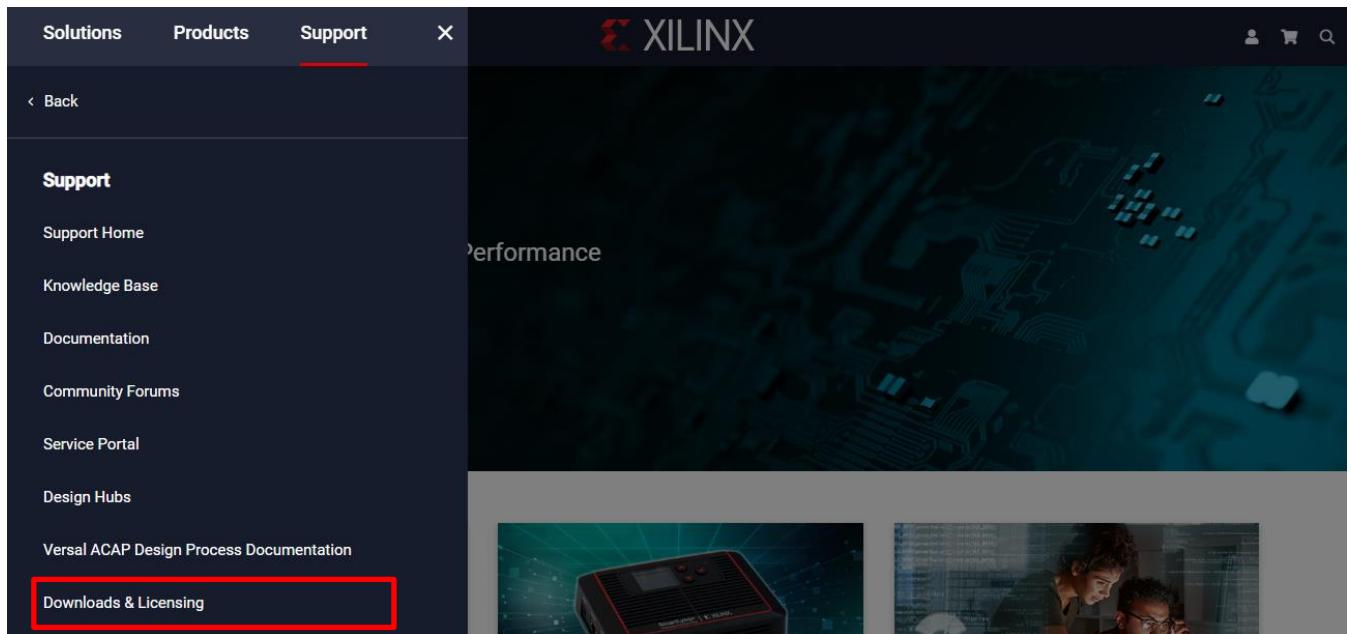


Figura 7 Ubicación de las Descargas en la página

En la Figura 7 se resalta la ubicación de la sección de descargas y licencias, al dar clic en ella se cargará la página que se muestra en la Figura 8, en donde se descargaría Vivado (HW Developer) en su versión 2020.2.

Vivado (HW Developer)	Vitis (SW Developer)	Vitis Embedded Platforms	Alveo Packages	PetaLinux	Device Models	Documentation Navigator
-----------------------	----------------------	--------------------------	----------------	-----------	---------------	-------------------------

Version						
2020.2						
2020.1						
2019.2						
Vivado Archive						
ISE Archive						
CAE Vendor Libraries Archive						

Vivado Design Suite - HLx Editions - 2020.2 Full Product Installation

Important Information	Download Includes	Vivado Design Suite HLx Editions (All Editions)
Vivado Design Suite 2020.2 is now available	Download Type	Full Product Installation
<ul style="list-style-type: none"> Public access support for the Xilinx Versal Platforms Petalinux now a part of Xilinx Unified Installer Access Block Design container now to create team-based designs Abstract Shell for Dynamic Function eXchange 2020.2 Introduces Vitis HLS for Vivado flows Add-on for MATLAB and Simulink (Unified Model Composer and System Generator) 	Last Updated	Nov 24, 2020
	Answers	2020.x - Vivado Known Issues
	Documentation	Release Notes OS Support Update What's New in Vivado

Figura 8 Versión a descargar (2020.2).

De las tres opciones de descargas disponible, se recomienda el instalador web autoextraíble, debido a que el desarrollo de las practicas se realizar en el Sistema Operativo Windows no se abarcara las consideraciones a tomar en Linux.

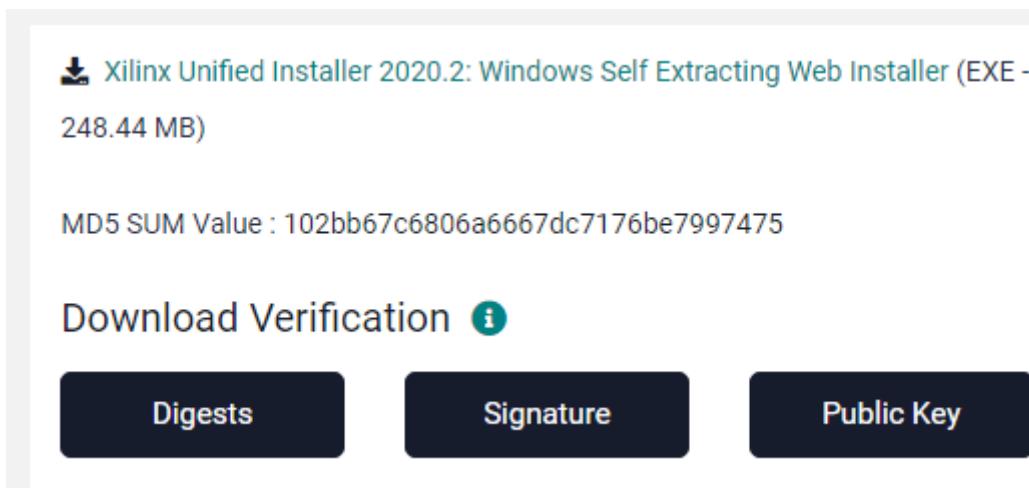


Figura 9 Instalador web autoextraíble (Windows).

Después de descargar el archivo, este se ejecuta como administrador, posteriormente se desplegará la ventana que aparece en la Figura 10 ,en “Preferences>Cores/CPUs Setting” se puede seleccionar la cantidad de ancho de banda de la CPU que se desea utilizar.

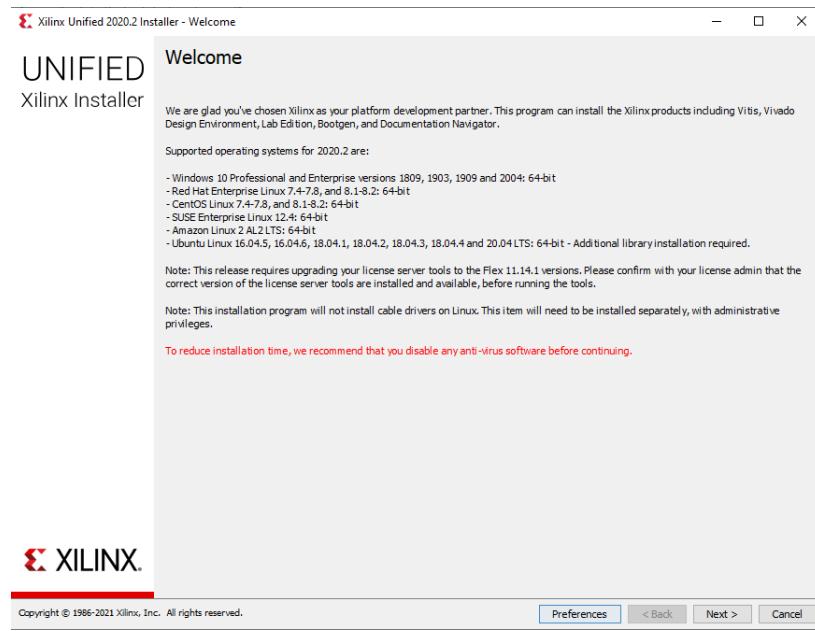


Figura 10 Bienvenida del asistente de instalación.

Después de dar clic en “Next>” se tendrá que iniciar sesión y tener seleccionado la opción “Download and Install Now” (ver Figura 11).

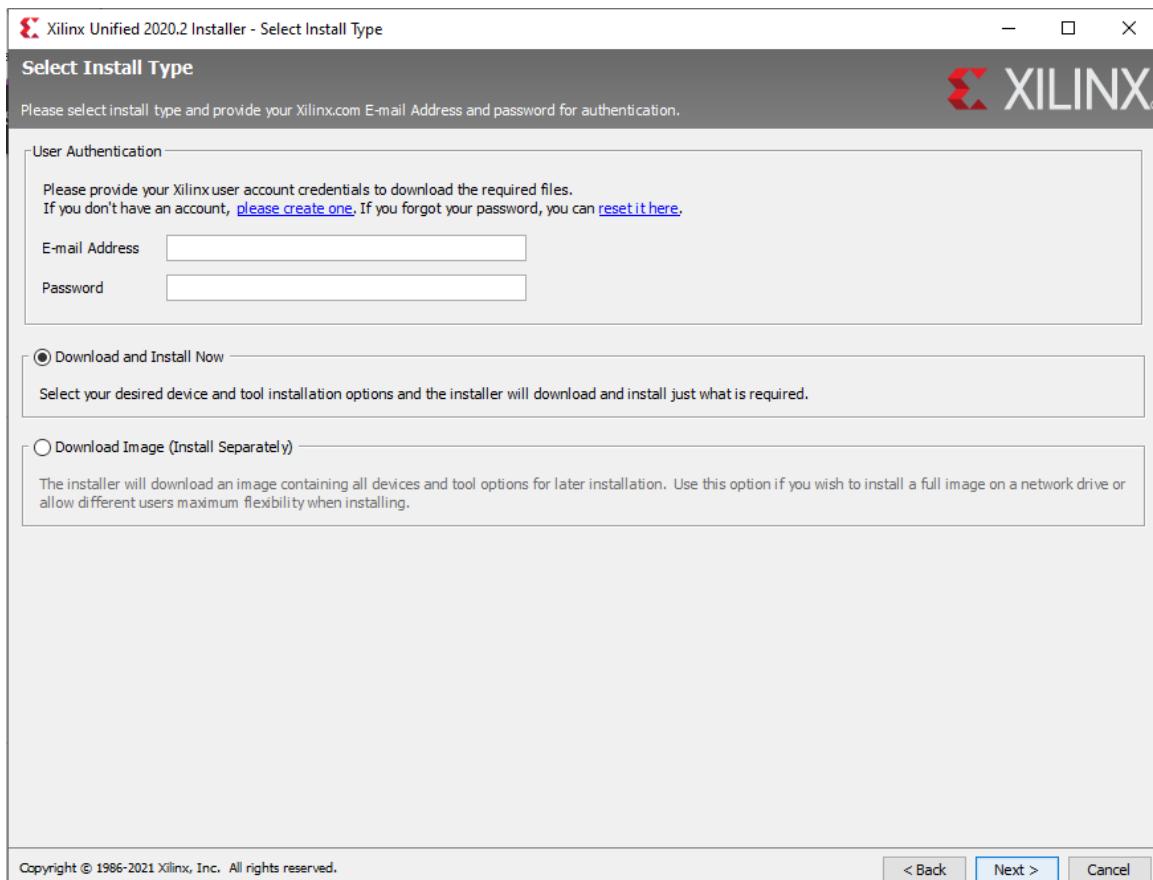


Figura 11 Inicio de sesión y tipo de descarga.

Después se desplegará la ventana que se aprecia en la Figura 12, en donde se debe seleccionar el producto que se instalará, se debe seleccionar Vitis y posteriormente dar clic en “Next>”

Nota:

El espacio requerido para la instalación al momento de realizar el manual fue de 130.68 GB.

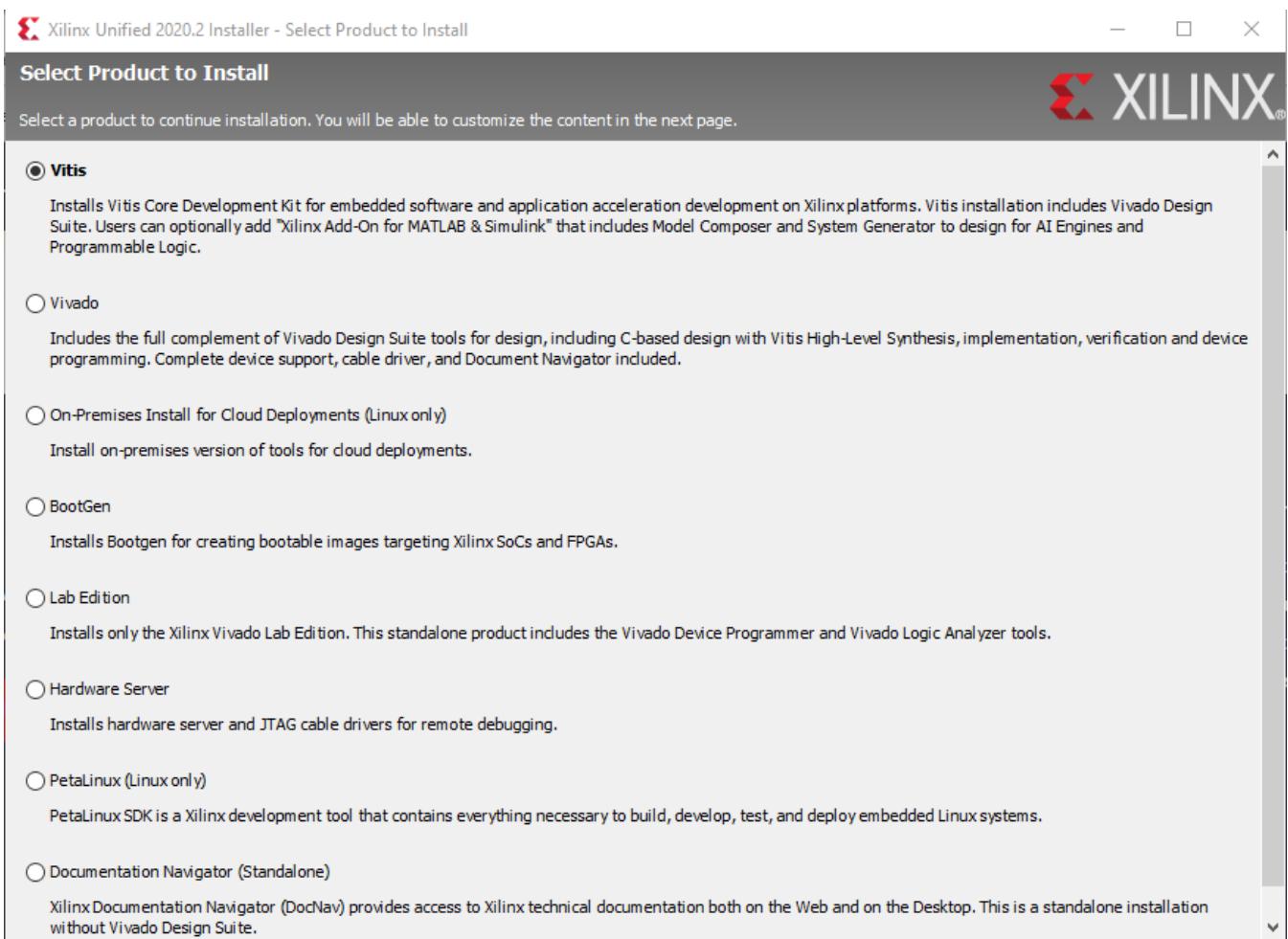


Figura 12 Selección del producto a Instalar.

En la siguiente ventana (ver Figura 13) se puede personalizar la instalación la recomendación es dejar lo que viene por default y dar clic en “Next>”, posteriormente se tiene que aceptar todos los acuerdos de licencia y dar clic en “Next>”.

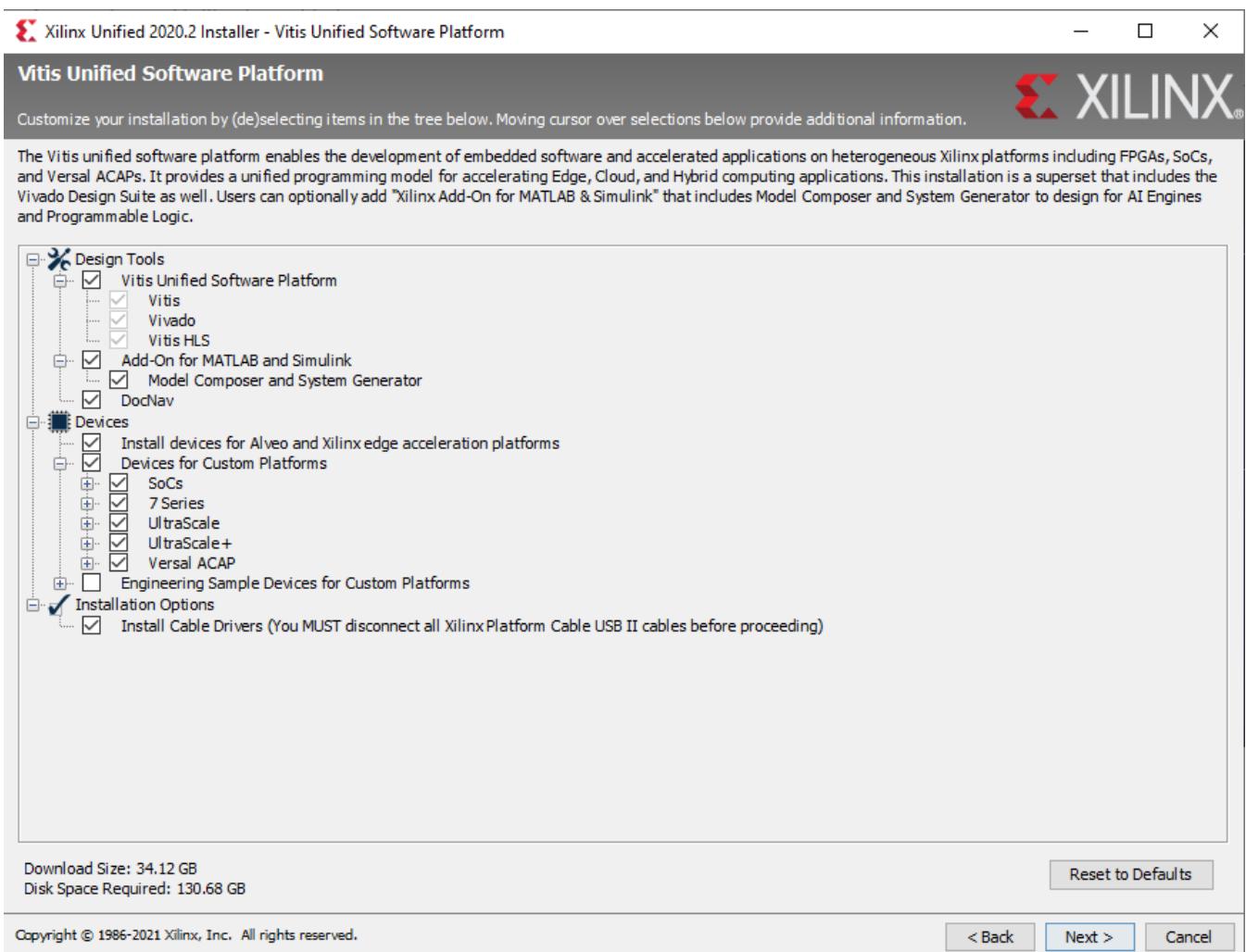


Figura 13 Personalizar instalación de Vitis.

Después se tiene que seleccionar la ubicación de la instalación, decidir si se genera los atajos para las aplicaciones y si es una instalación para todos los usuarios de la computadora, una vez configurado esto, se da clic en “Next>” y comienza la descarga-instalación.

Nota: En la Figura 14 aparecen algunas advertencias, esto debido a que el software ya se encontraba instalado.

Nota: Si se cuenta con una conexión a internet intermitente es importante estar pendiente de la descarga, ya que usualmente suelen surgir mensajes de error al descargar y es necesario dar clic en “Retry” para que verifique los archivos ya descargados y reanude la descarga.

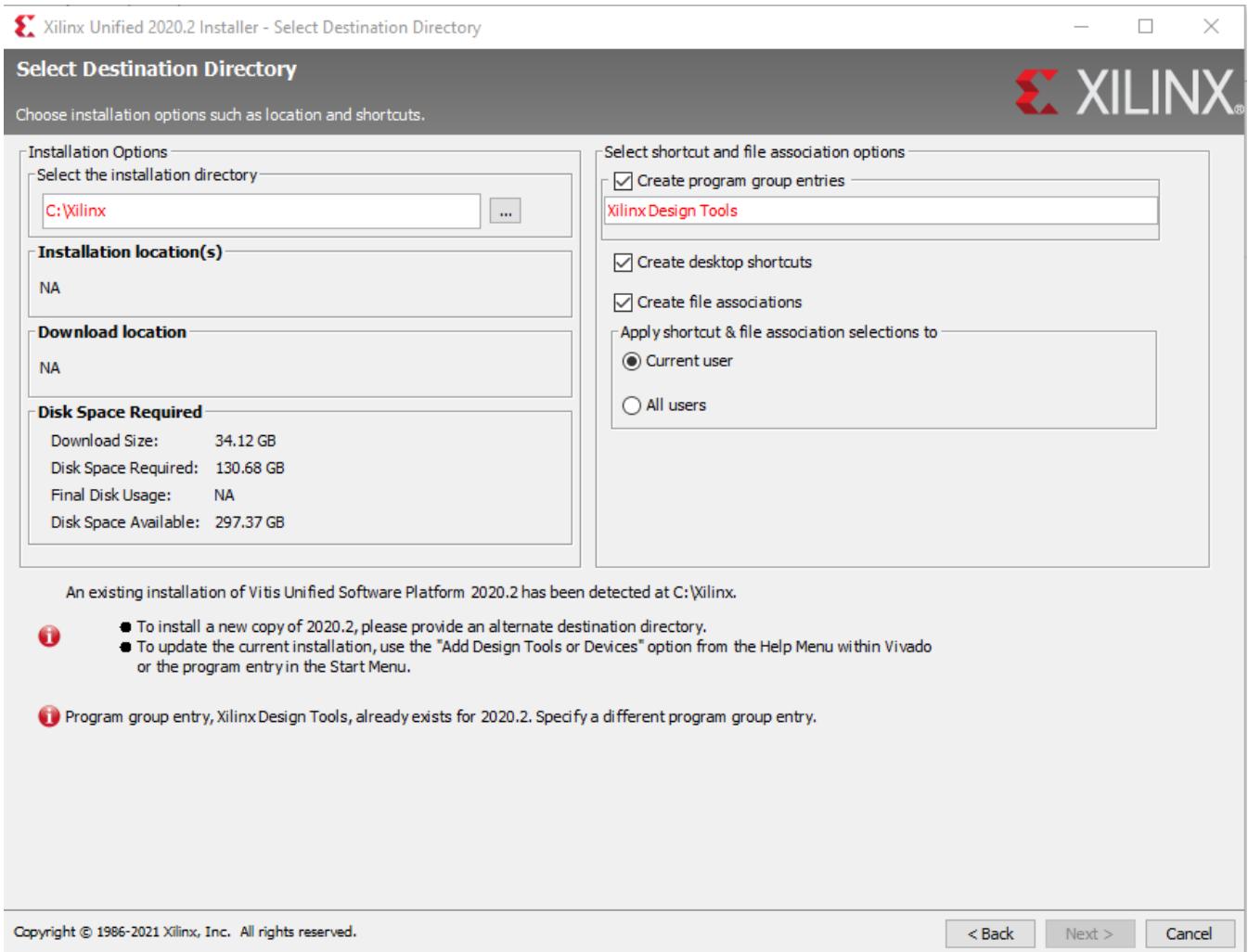


Figura 14 Ubicación de la instalación.

Preparativos para utilizar la tarjeta PYNQ Z2.

Para facilitar las cosas al momento de crear un proyecto es recomendable tener toda la documentación de la tarjeta PYNQ Z2, para esto se tiene que ir a la página del fabricante: <https://www.tul.com.tw/productspynq-z2.html>.



KU115

PYNQ™-Z2

BTU9P

BTU9P PRO

TUL PYNQ™-Z2 board, based on Xilinx Zynq SoC, is designed for the Xilinx University Program to support PYNQ (Python Productivity for Zynq) framework (please refer to the PYNQ project webpage at www.pynq.io) and embedded systems development.

[TUL PYNQ-Z2 Product Specification \(PDF\)](#)



Product Specification

Figura 15 Pagina del fabricante.

Instalación de la tarjeta.

Para descargar los archivos correspondientes a la tarjeta, se tiene que localizar la sección de descargas (ver Figura 16), posteriormente al hacer clic en “PYNQ-Z2 Board File” comenzara la descarga de un archivo con nombre “pynq-z2.zip”.

Downloads

- PYNQ-Z2 User Manual (PDF)
- PYNQ-Z2 Boot Image
 - 1. V2.3
 - 2. V2.4
 - 3. V2.5
- PYNQ-Z2 Board File (for Pmod IP support please refer [here](#))
- Master XDC
- Protective Acrylic Case (PDF)
- Zynq Datasheet (PDF)
- Zynq Manual (PDF)
- Schematics (PDF)

Figura 16 Hipervínculo para descargar los archivos de la tarjeta.

La carpeta que se encuentra dentro del archivo comprimido se copia y se pega en la siguiente ruta “C:\Xilinx\Vivado\2020.2\data\boards\board_files” (ver).

Este equipo > Disco local (C:) > Xilinx > Vivado > 2020.2 > data > boards > board_files					Buscar en board_files
Nombre	Fecha de modificación	Tipo	Tamaño		
kc705	19/01/2021 08:12 p. m.	Carpeta de archivos			
kcu105	19/01/2021 08:00 p. m.	Carpeta de archivos			
kcu116	19/01/2021 08:01 p. m.	Carpeta de archivos			
kcu1500	19/01/2021 08:07 p. m.	Carpeta de archivos			
li-imx274-mipi	19/01/2021 08:16 p. m.	Carpeta de archivos			
nexys_video	21/01/2021 12:07 p. m.	Carpeta de archivos			
nexys4	21/01/2021 12:07 p. m.	Carpeta de archivos			
nexys4_ddr	21/01/2021 12:07 p. m.	Carpeta de archivos			
nexys-a7-50t	21/01/2021 12:07 p. m.	Carpeta de archivos			
nexys-a7-100t	21/01/2021 12:07 p. m.	Carpeta de archivos			
pynq-z2	21/01/2021 10:28 p. m.	Carpeta de archivos			
sp701	19/01/2021 08:11 p. m.	Carpeta de archivos			
sword	21/01/2021 12:07 p. m.	Carpeta de archivos			
usb104-a7	21/01/2021 12:07 p. m.	Carpeta de archivos			
vc707	19/01/2021 08:11 p. m.	Carpeta de archivos			
vc709	19/01/2021 08:11 p. m.	Carpeta de archivos			
vck190	19/01/2021 08:14 p. m.	Carpeta de archivos			

Figura 17 Ubicación final de la carpeta pynq-z2

Crear Proyecto en Vivado

Al abrir Vivado 2020.2 se desplegará la ventana que aparece en la Figura 18.

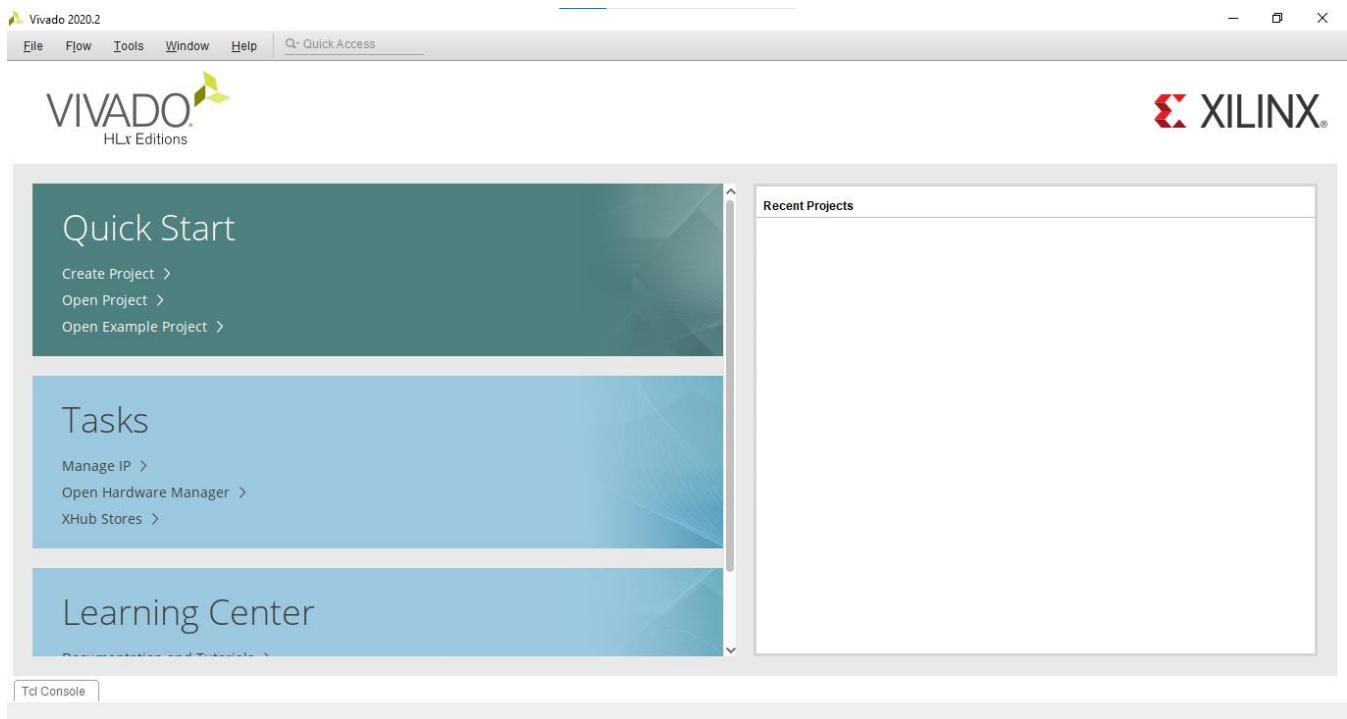


Figura 18 Ventana de inicio de Vivado.

Al hacer clic en “Create Project” se desplegará un asistente para ayudar a la creación del proyecto.

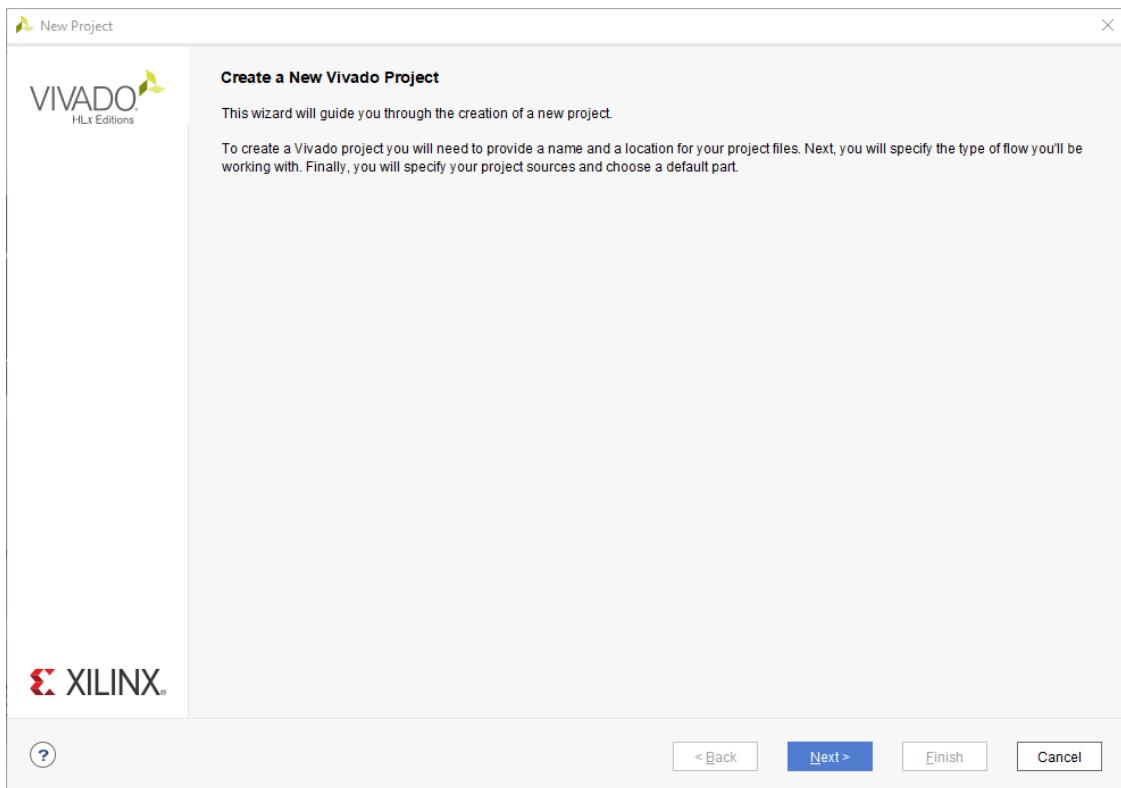


Figura 19 Asistente para la creación del proyecto.

Posteriormente se tiene que escoger un nombre y especificar la ubicación donde se creara el proyecto, también se puede decidir si se crea un subdirectorio para el proyecto (ver Figura 20).

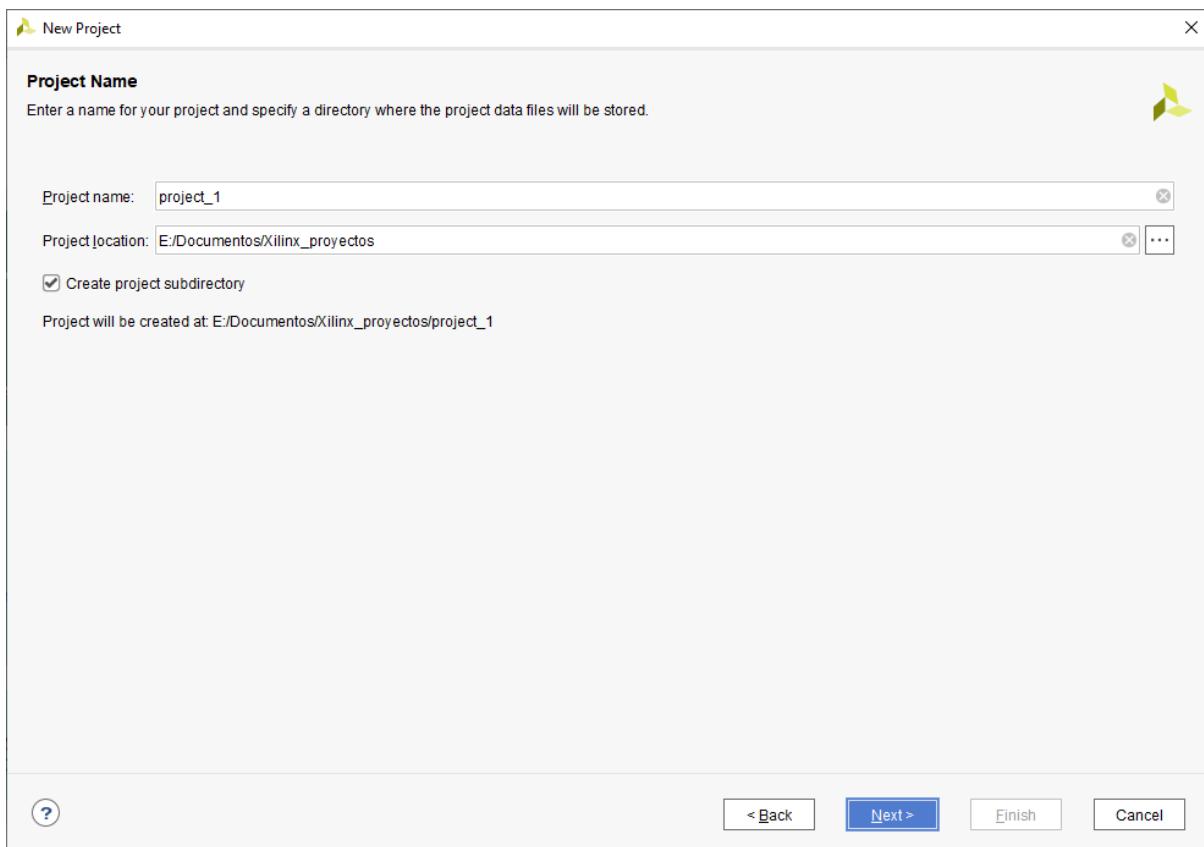


Figura 20 Nombre del proyecto y ubicación.

En la siguiente ventana, se tiene que escoger el tipo del proyecto, dado las características, siempre se trabajara con el “RTL Project”, se puede especificar archivos fuentes y tambien decidir si se hace extensible a la plataforma de Vitis. (ver Figura 21).

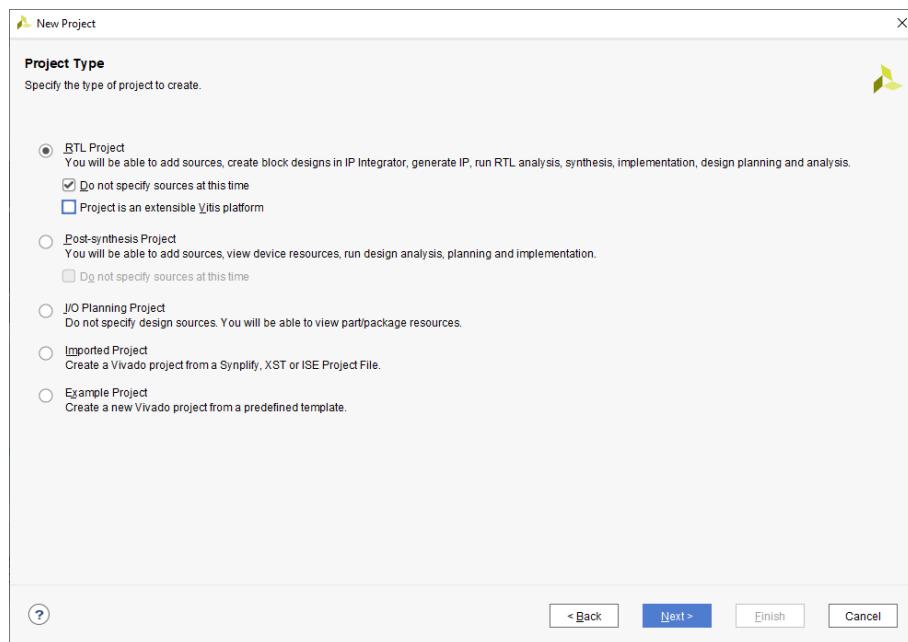


Figura 21 Tipo de proyecto (Vivado).

Posteriormente, se tiene que especificar la tarjeta a utilizar o el componente, dado que ya se realizó la instalación de la tarjeta, en la pestaña “Boards” únicamente se realizará la búsqueda y selección de la tarjeta (ver Figura 22).

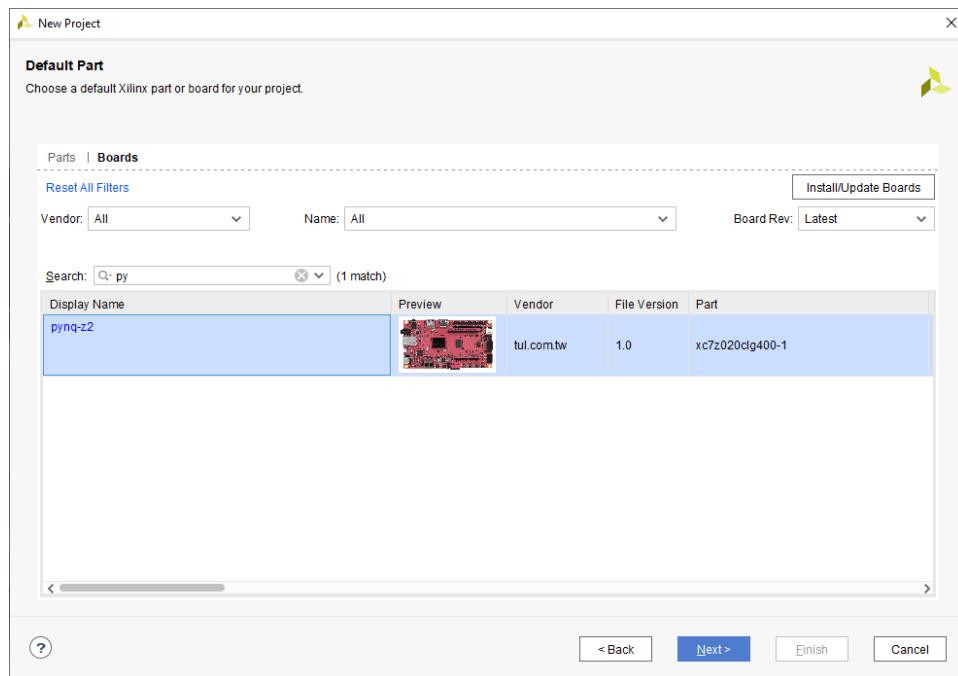


Figura 22 Selección de la tarjeta.

Finalmente, en la última ventana se muestra un resumen del proyecto, se revisa que todo se haya especificado de manera correcta y se da clic en “Finish”. (ver Figura 23).

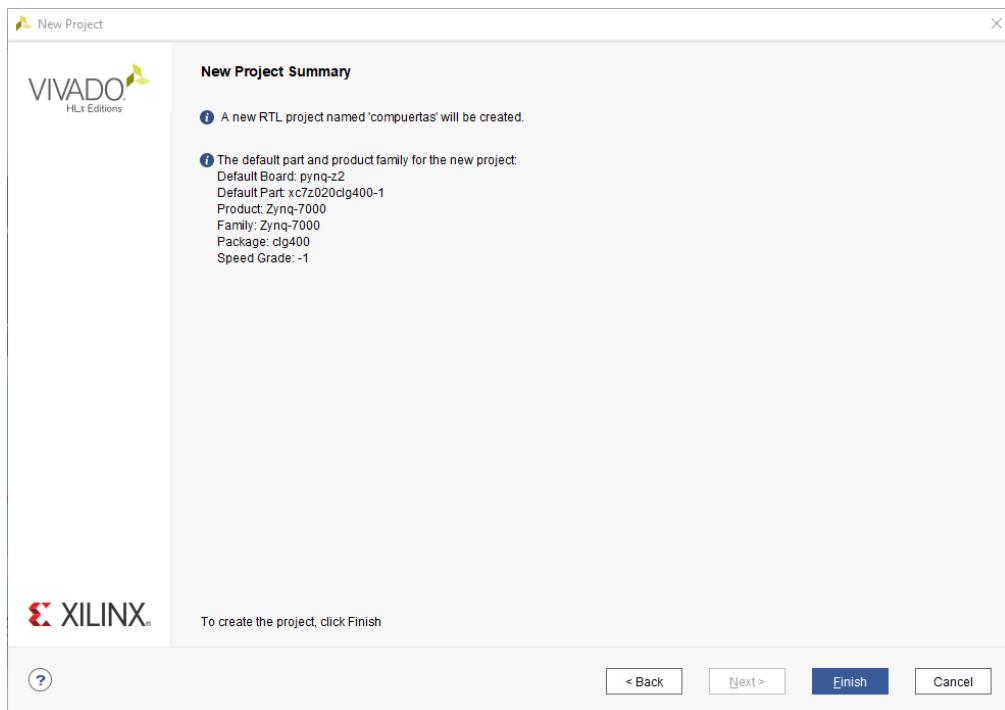


Figura 23 Resumen del proyecto.

Añadir archivos al proyecto.

En la ventana “Source” se despliegan todos los archivos correspondientes al proyecto, los archivos se encuentran clasificados en grupos:

- Design Sources.
- Constraints.
- Simulation Sources.
- Utility Sources.

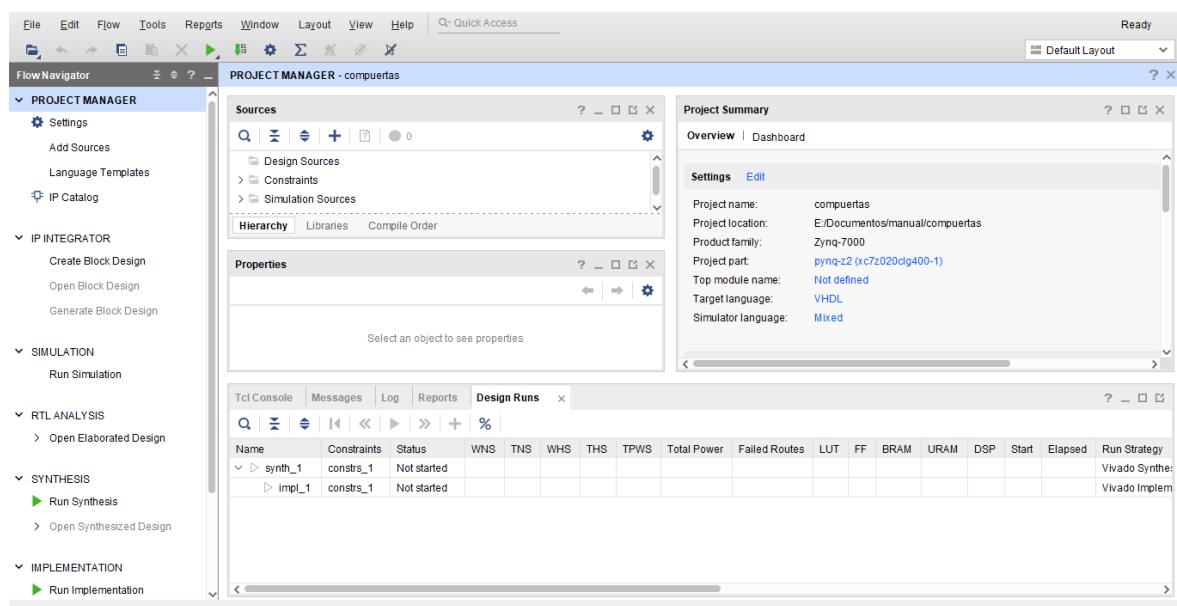


Figura 24 Proyecto en Vivado.

De momento únicamente se hará referencia a los primeros 3 grupos. Para añadir un archivo de diseño se tienen dos opciones:

- Situarse sobre una parte en blanco de la ventana y dar clic derecho > "Add Sources..."
- Dando clic directamente al icono +.

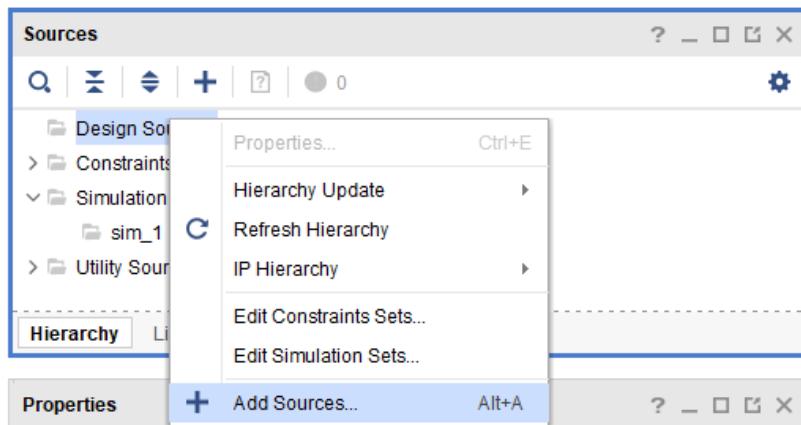


Figura 25 Añadir archivos al proyecto.

De cualquier manera, se desplegará la siguiente ventana que aparece en la Figura 26, en donde se debe seleccionar el tipo de archivo que se va a agregar.

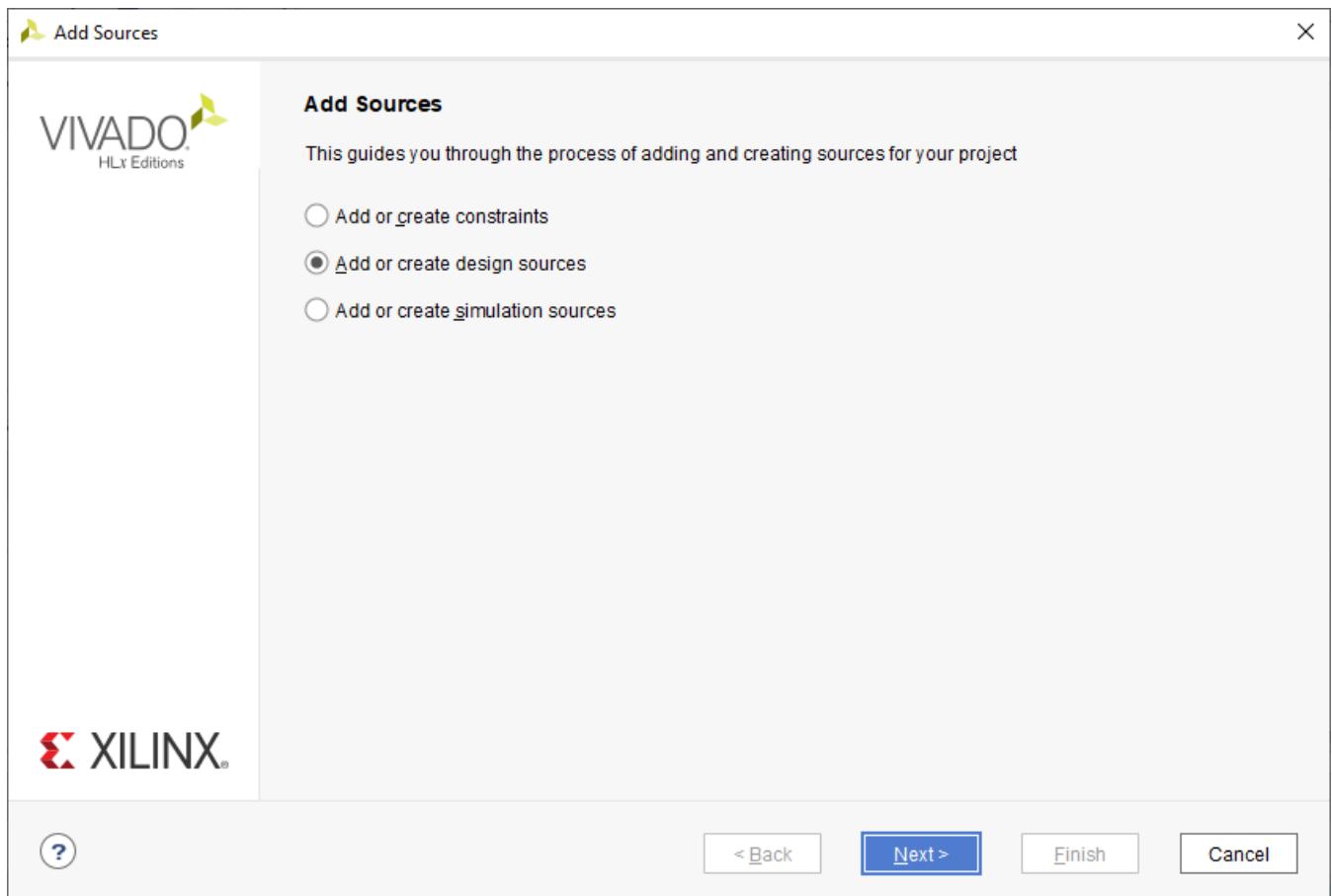


Figura 26 Asistente para añadir archivos.

De manera general se tienen 3 opciones (ver Figura 27):

- Add Files.
- Add Directories. (Esta opción no está disponible en Constraints Source).
- Create File.

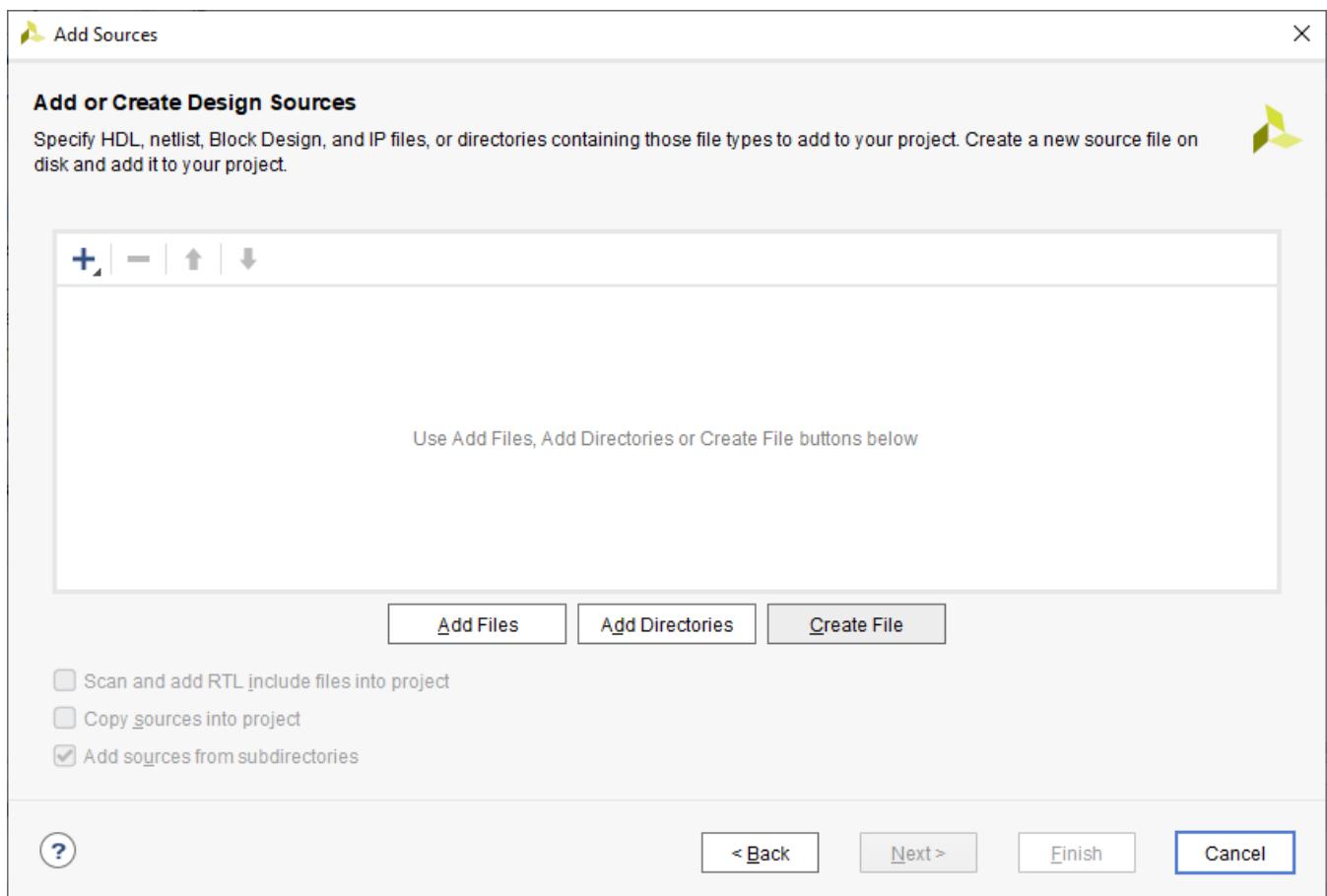


Figura 27 Añadir-Crear archivos.

En las dos primeras opciones, se despliega una ventana donde se tiene que proporcionar la ubicación del archivo o directorio a añadir (ver Figura 28), en ambos casos se puede optar por copiar los archivos al proyecto, lo cual es recomendable.

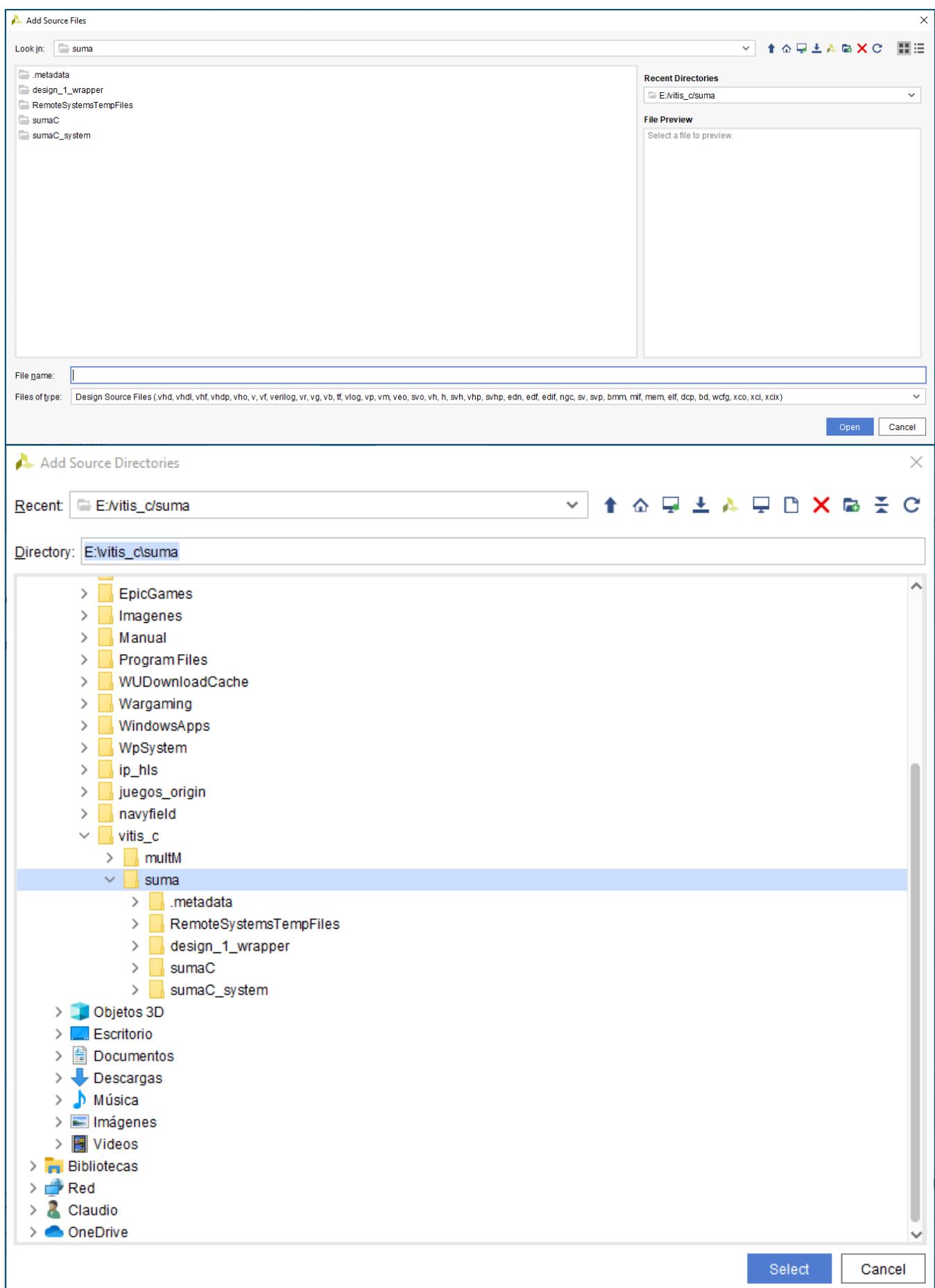


Figura 28 Ubicación del archivo o directorio.

En el caso quiera añadir un archivo nuevo, el proceso cambia:

- Design Source – Simulation Source

Para ambos casos se tiene que especificar:

- Tipo de archivo.
 - Verilog.
 - Verilog Header.
 - SystemVerilog.
 - VHDL.
 - Memory File.
- Nombre del archivo.
- Ubicación del archivo.

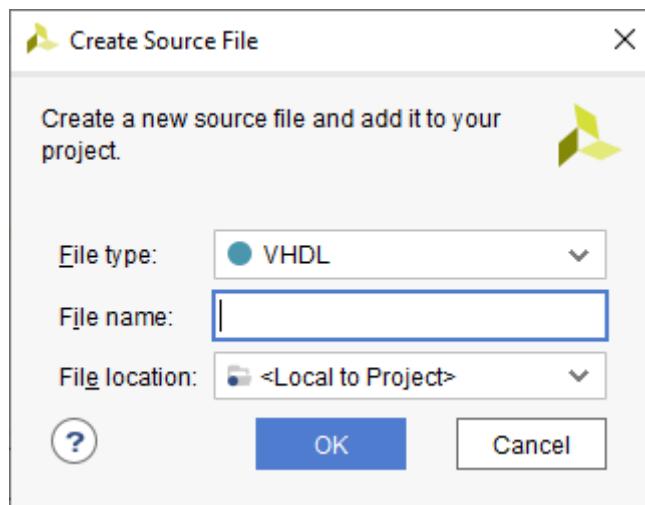


Ilustración 1 Creación de archivo Fuente.

- Constraints Source.

De igual forma para los archivos constraints se tienen que especificar:

- Tipo de archivo.
 - XDC
- Nombre del archivo.
- Ubicación del archivo.

NOTA:

Para obtener el archivo XDC correspondiente a la tarjeta PYNQ-Z2, la forma más sencilla es ir a la página del fabricante (<https://www.tul.com.tw/productspynq-z2.html>) y descargar el archivo “Master XDC” (ver).

Downloads

- PYNQ-Z2 User Manual (PDF)
- PYNQ-Z2 Boot Image
 - 1. V2.3
 - 2. V2.4
 - 3. V2.5
- PYNQ-Z2 Board File (for Pmod IP support please refer here)
- Master XDC

Figura 29 Descarga del archivo XDC.

Al dar clic, se descarga un archivo con nombre “pynq-z2_v1.0.xdc.zip”, dentro del archivo comprimido, se encuentran los siguientes archivos:

 PYNQ-Z2 v1.0.tcl	Archivo TCL	5 KB	No
 PYNQ-Z2 v1.0.xdc	Archivo XDC	3 KB	No

Figura 30 Archivos dentro de comprimido pynq-z2_v1.0.xdc

Se tienen que quitar el comentario de la sección a utilizar.

Síntesis – Implementación -Bitstream.

Una vez que se haya terminado el proyecto para realizar la Síntesis, Implementación o Generación del Bitstream, se realizan mediante la ventana de “Flow Navigator” (ver Figura 32).

Adicionalmente para los 3 casos se pueden realizar desde la barra de herramientas.(ver Figura 31)

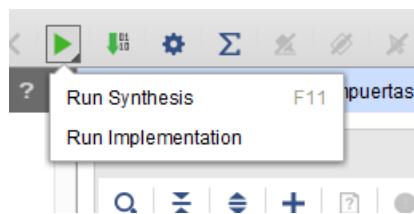


Figura 31 Síntesis Implementación y Bitstream en la barra de herramientas.

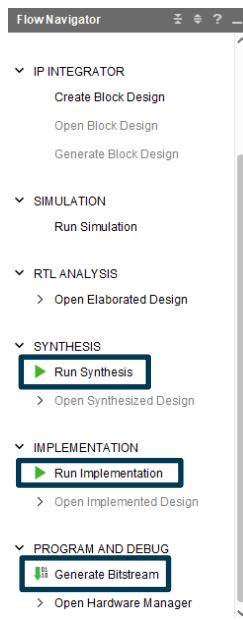


Figura 32 Opciones de Síntesis, Implementación y Bitstream.

Práctica 1: Zynq con módulo RTL

Objetivos

- Crear un módulo RTL.
- Diseñar un diagrama donde se comunique el módulo RTL con el SoC Zynq.
- Crear una aplicación en el Software Development Kit (SDK).

Procedimiento

El proceso para realizar la práctica se divide en 2 secciones, la parte referente al diseño del hardware que se realiza en Vivado y lo referente a la aplicación, la cual se puede realizar en el SDK o en Jupyter.



Desarrollo

Vivado

Para la parte de Vivado, se crea un nuevo proyecto, para este en particular no se especificarán archivos. Por lo que el primer paso es crear un archivo de diseño, el cual estará en lenguaje VHDL. Este será el módulo RTL, anteriormente mencionado. La descripción se encuentra en el Código 1.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity gates_mux is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : out STD_LOGIC;
           sel : in STD_LOGIC_VECTOR (1 downto 0));
end gates_mux;

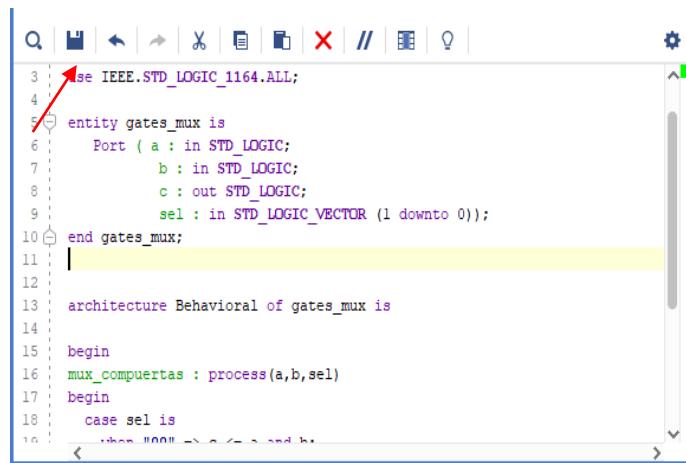
architecture Behavioral of gates_mux is

begin
    mux_compuertas : process(a,b,sel)
    begin
        case sel is
            when "00" => c <= a and b;
            when "01" => c <= a or b ;
            when "10" => c <= a xor b;
            when others => c <= a nor b ;
        end case;
    end process mux_compuertas;

end Behavioral;
```

Código 1 Descripción en VHDL del multiplexor.

Después de la modificación, se guarda el archivo, esto puede realizarse dando clic en el ícono (ver Figura 33) o utilizando el atajo CTRL+S.



```
3 use IEEE.STD_LOGIC_1164.ALL;
4
5 entity gates_mux is
6     Port ( a : in STD_LOGIC;
7            b : in STD_LOGIC;
8            c : out STD_LOGIC;
9            sel : in STD_LOGIC_VECTOR (1 downto 0));
10 end gates_mux;
11
12
13 architecture Behavioral of gates_mux is
14 begin
15     mux_compuertas : process(a,b,sel)
16     begin
17         case sel is
18             when "00" => c := a and b;
```

Figura 33 Guardar modificaciones en archivos.

Posteriormente se crea un diseño de bloque, esto se puede realizar desde Flow Navigator (ver Figura 34).

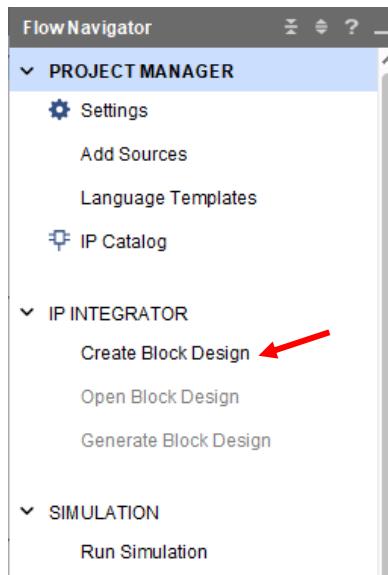


Figura 34 Crear bloque Diseño de bloque.

Dado que en este proyecto únicamente se utilizará un diseño de bloque se deja el nombre por default. (ver Figura 35)

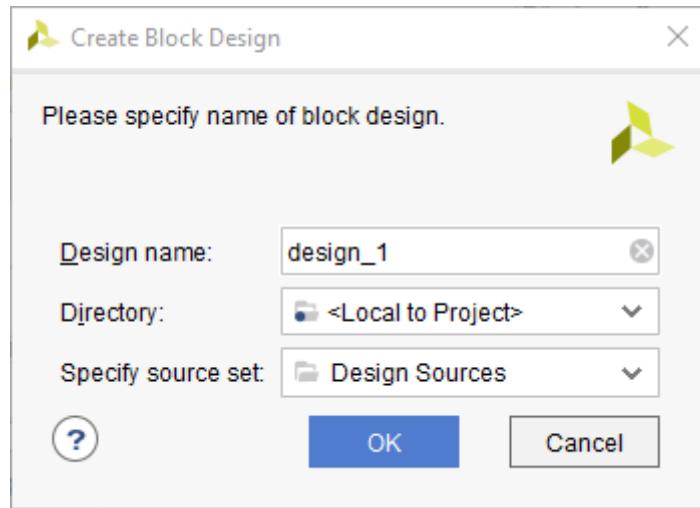


Figura 35 Creación del bloque de diseño.

Una vez creado el bloque de diseño, se tiene que agregar el módulo RTL, esto se hace haciendo clic derecho sobre él y eligiendo la opción “Add module to Block Design” (ver Figura 36).

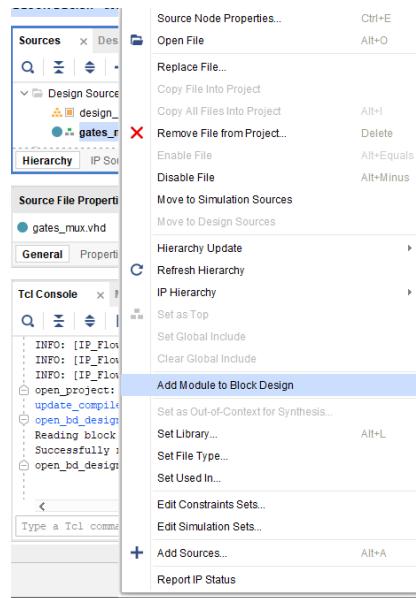


Figura 36 Agregar modulo al diseño de bloque.

Una vez realizado esto, en el diagrama del block Design aparecerá el módulo RTL, como se aprecia en la Figura 37.

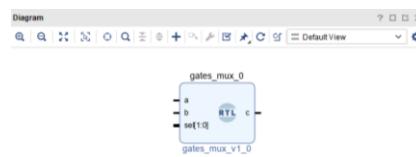


Figura 37 Block Design del proyecto.

Después de esto se tiene que añadir las siguientes IPs:

- Zynq Processor.

- GPIO – 2 modulos.

Esto se realiza dando clic derecho sobre un espacio en blanco del diagrama y seleccionando “Add IP” (ver Figura 38) o utilizando el atajo Ctrl+I.

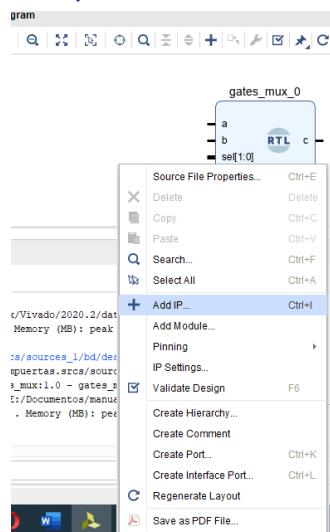


Figura 38 Agregar IP.

Una vez agregado los 2 IPs, el diagrama será similar al de la Figura 39, para que se “acomoden” los bloques se puede utilizar la herramienta “Regenerate layout” o acomodarlos manualmente.

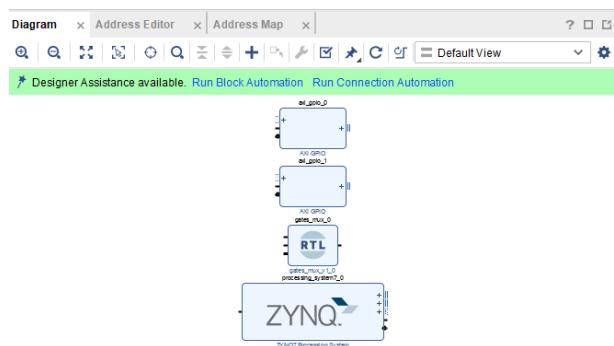


Figura 39 Diagrama con los 4 módulos.

En la Figura 39 se puede apreciar que la asistencia de diseño está disponible con dos opciones:

- Run Block Automation
- Run connection Automation.

El primero que se usara es el “Run Block Automation”, una vez hecho clic sobre él, se abrirá una ventana igual a la que aparece en la Figura 40, en este caso se deja con las opciones por defecto y se da clic en “OK”.

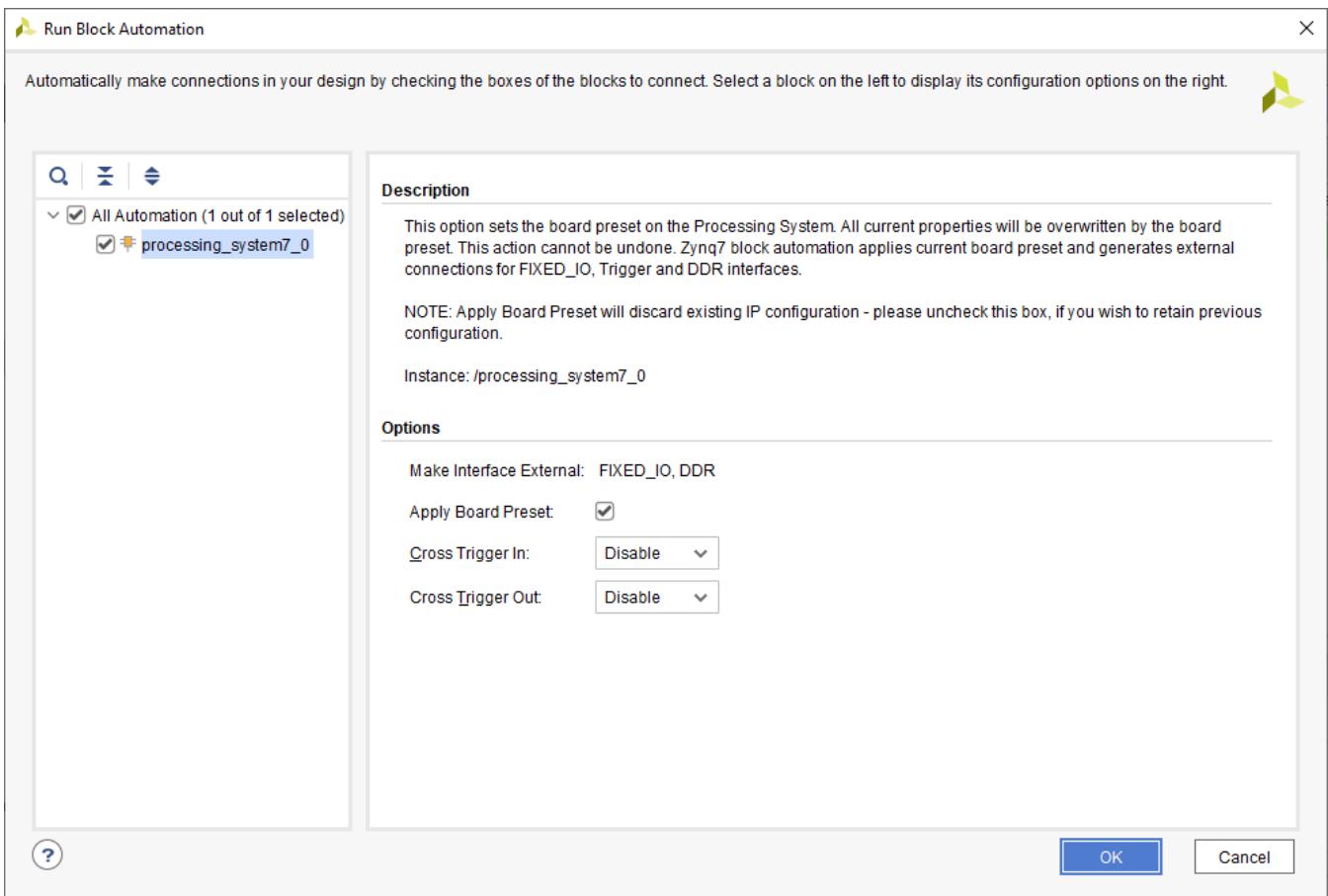


Figura 40 Configuración de Run Block Automation.

La forma en que se utilizara las dos GPIO es la siguiente:

- GPIO_0: la salida del módulo RTL (c).
- GPIO_1: la entrada de selección del bloque RTL(Sel).

El siguiente paso es personalizar al GPIO, esto se realiza dando doble clic izquierdo sobre él o dando clic derecho y seleccionar “Customize Block”, en cualquiera de las dos formas se desplegará la ventana que aparece en la Figura 41.

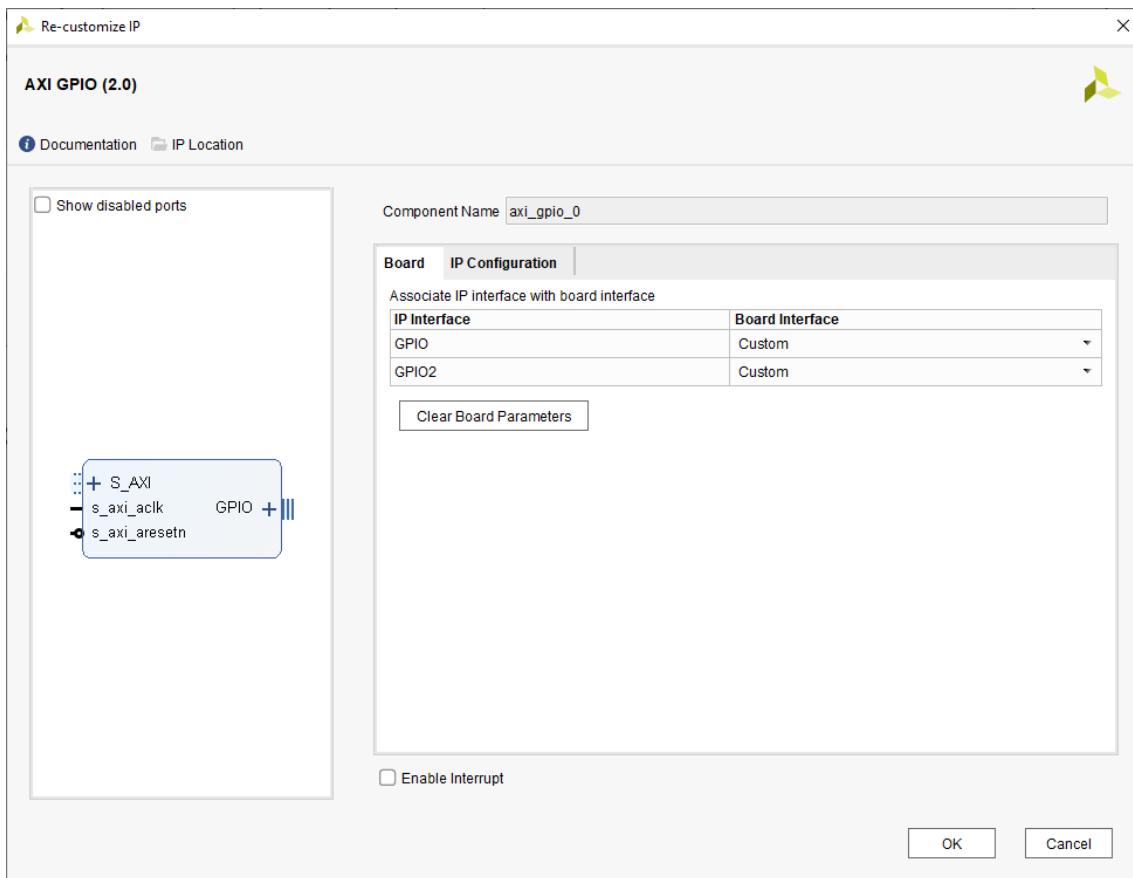


Figura 41 Ventana para personalizar la IP.

Tanto para la selección y salida se debe tener seleccionada la opción “Custom” en “Board Interface”, en ambas, únicamente se utilizará un canal.

Nota: En “Board interface” se pueden seleccionar las diferentes entradas-salidas disponibles en la tarjeta (ver Figura 42), al hacer esto ya no es necesario especificarlas en el archivo XDC.

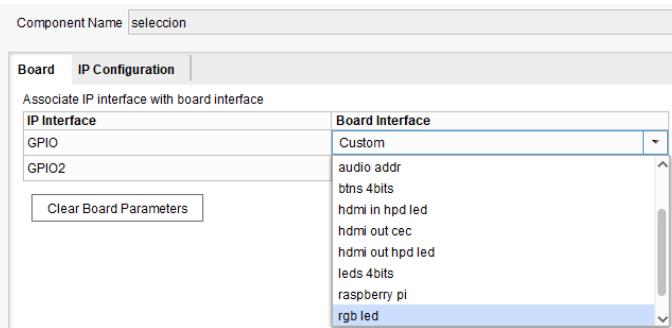


Figura 42 Board Interface.

Después de configurar correctamente los parámetros de la pestaña “Board”, se realizan los cambios necesarios en la pestaña “IP Configuration” como se muestra en la Figura 43.

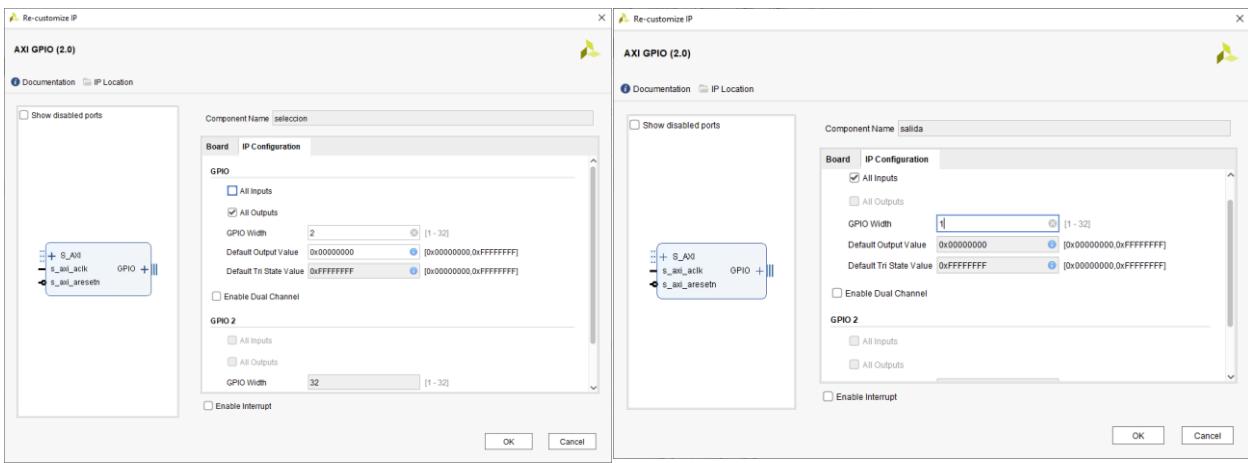


Figura 43 IP Configuración de ambos módulos, selección (izquierda) y salida(derecha).

Despues de realizar esta configuracion, se tienen que realizar las conexiones como se muestra en la Figura 44.

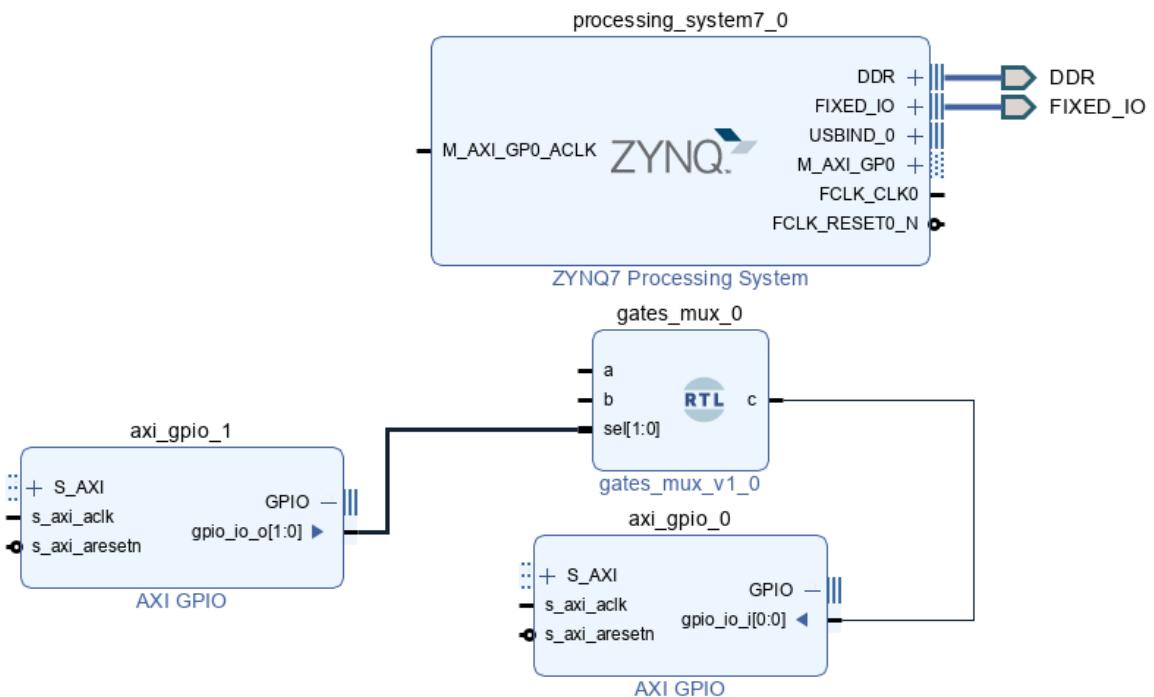


Figura 44 Diagrama de conexión de los módulos.

El siguiente paso es hacer externas las entradas (a y b) y salida (c) del módulo RTL, esto se realiza haciendo clic derecho sobre una terminal y seleccionando “Make External” o seleccionando la terminal y usar el atajo Ctrl+T (ver Figura 45).

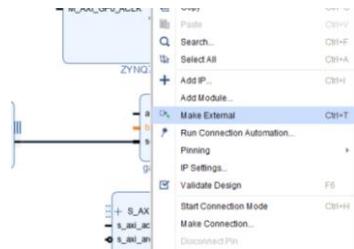


Figura 45 Hacer externo una terminal del diagrama.

Una vez hecho esto, el diagrama debe lucir como en la Figura 46.

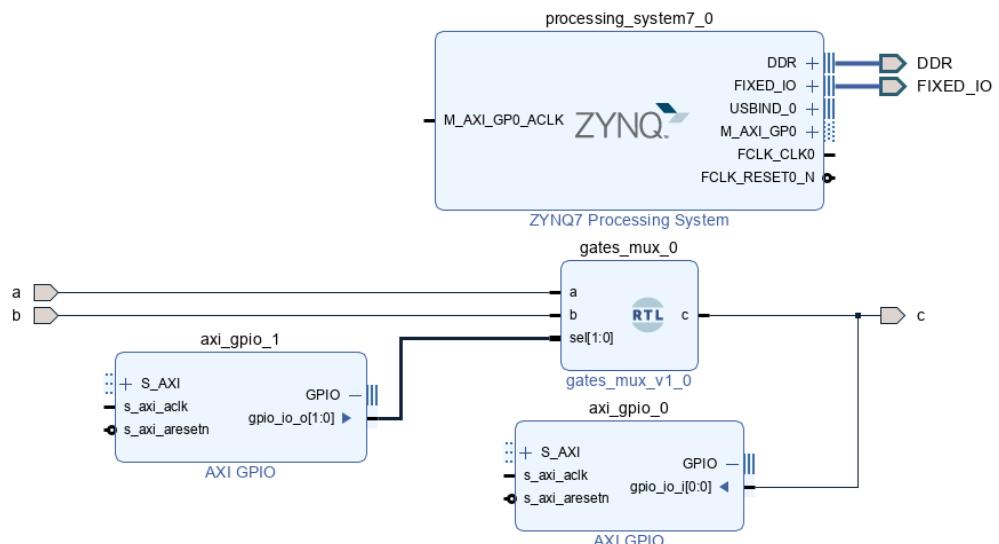
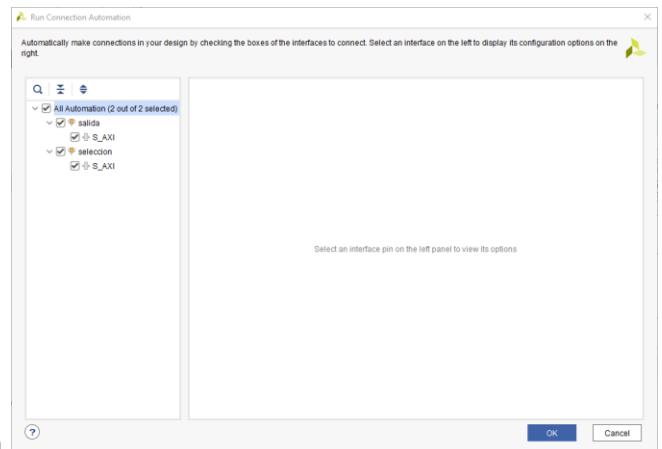


Figura 46 Diagrama con las terminales externas.

Una vez hecho esto, se utiliza la asistencia “Run Connection Automation”, al hacer clic se



desplegará la ventana que se muestra en la Figura 47, se tiene que marcar la casilla “All Automation”.

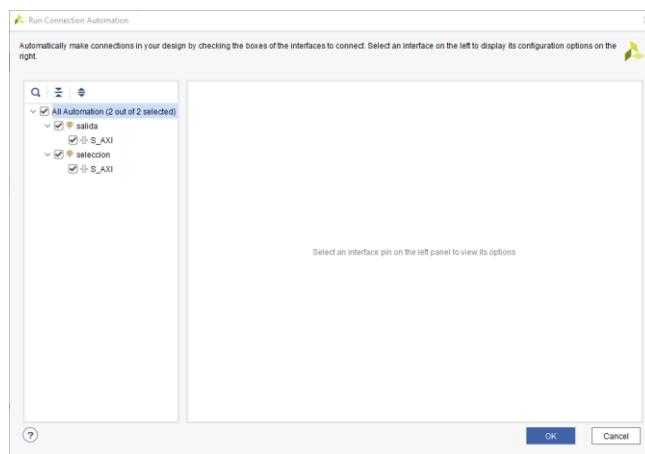


Figura 47 Ventana de la asistencia Run Automation Block

Una vez hecho esto, el diagrama será similar al de la Figura 48, se recomienda utilizar “Regenerate Layout” y “Optimize Routing”.

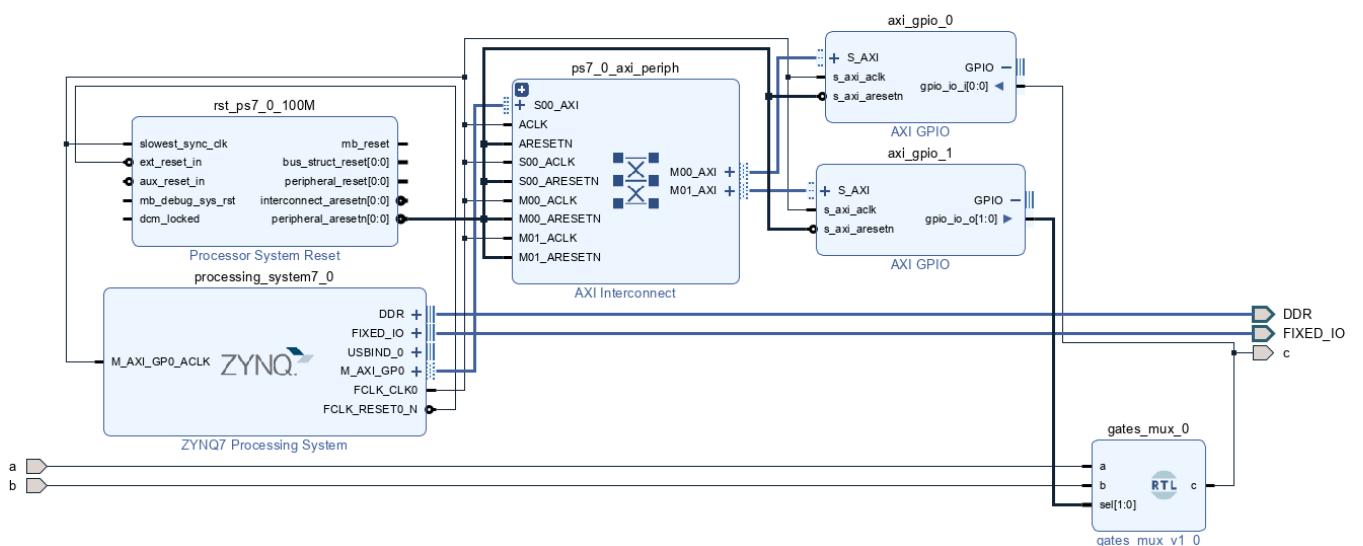


Figura 48 Diagrama final.

El paso siguiente es validar el diseño, esto se realizar dando clic sobre el icono (ver Figura 49) o tecleando F6.

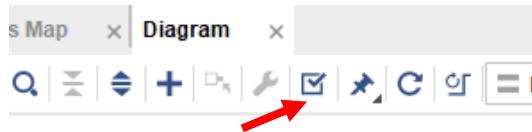


Figura 49 Icono Validate Design

En caso de no haber error, se desplegará el cuadro de la Figura 50.

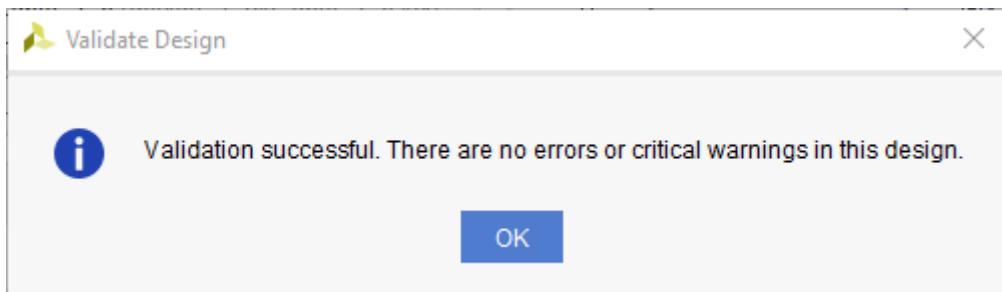


Figura 50 Validación exitosa.

Posteriormente se añade el archivo XDC, se borran los “#” para quitar el comentario y colocar el nombre de los puertos externos correspondientes, en este caso se optó por utilizar los switches como las entradas y un led como la salida (ver Figura 51).

```
compuertas.xdc*
E:\Documentos\manual\compuertas\compuertas.scs\contra_1\new\compuertas.xdc
Q | H | < | > | X | // | ? |
11: ##Switches
12:
13: #set_property -dict { PACKAGE_PIN M20 IOSTANDARD LVCMOS33 } [get_ports { a[1] }; #IO_L2P_T1_ADN3_35 Sch=mr{0}
14: #set_property -dict { PACKAGE_PIN M19 IOSTANDARD LVCMOS33 } [get_ports { b[1] }; #IO_L2P_T1_ADP2_35 Sch=mr{1}
15:
16: ##RGB LEDs
17:
18: #set_property -dict { PACKAGE_PIN L15 IOSTANDARD LVCMOS33 } [get_ports { led4_p[1] }; #IO_L2P_T2_ADM1_35 Sch=led4_b
19: #set_property -dict { PACKAGE_PIN G17 IOSTANDARD LVCMOS33 } [get_ports { led4_g[1] }; #IO_L2P_T2_36_Sch=led4_g
20: #set_property -dict { PACKAGE_PIN N16 IOSTANDARD LVCMOS33 } [get_ports { led4_b[1] }; #IO_L2P_T2_DQS_ADI4P_35 Sch=led4_
21: #set_property -dict { PACKAGE_PIN G14 IOSTANDARD LVCMOS33 } [get_ports { led5_p[1] }; #IO_0_35_Sch=led5_b
22: #set_property -dict { PACKAGE_PIN L14 IOSTANDARD LVCMOS33 } [get_ports { led5_g[1] }; #IO_L2P_T2_ADV1_35 Sch=led5_g
23: #set_property -dict { PACKAGE_PIN M15 IOSTANDARD LVCMOS33 } [get_ports { led5_b[1] }; #IO_L2P_T2_36_Sch=led5_x
24:
25: ##LEDs
26:
27: #set_property -dict { PACKAGE_PIN B14 IOSTANDARD LVCMOS33 } [get_ports { led[1] }; #IO_L2P_T2_USBF_34_Sch=led0]
28: #set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 } [get_ports { led[1] }; #IO_L2P_T2_34_Sch=led1]
29: #set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports { led[1] }; #IO_L2P_T2_DQS_ADI4P_35_Sch=led2]
30: #set_property -dict { PACKAGE_PIN M14 IOSTANDARD LVCMOS33 } [get_ports { led[1] }; #IO_L2P_T2_36_Sch=led3]
31:
32: ##Bitstream
33:
```

Figura 51 Archivo XDC utilizado.

Es importante recalcar que se deben guardar las modificaciones realizadas a los archivos, a continuación, se crea el HDL Wrapper, esto se hace seleccionando el Block Design, clic derecho y se elige “Create HDL Wrapper” (ver Figura 52).

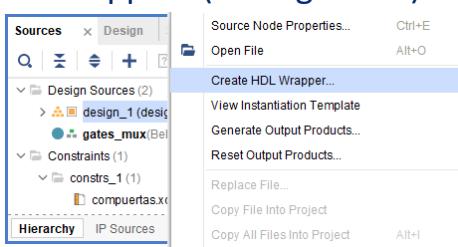


Figura 52 Crear HDL Wrapper

En seguida se genera el Bitstream, una vez finalizado el proceso se desplegará la ventana que aparece en la Figura 53.

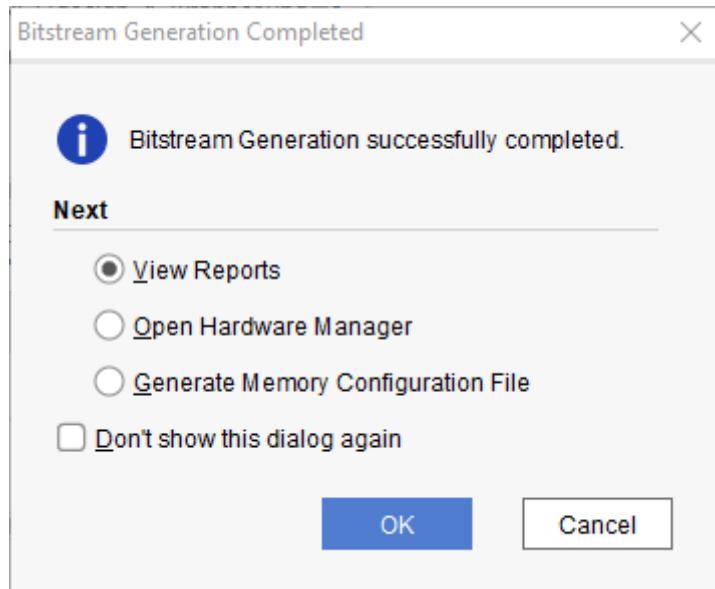


Figura 53 Generación exitosa del Bitstream

Finalmente se tiene que exportar el hardware a Vitis, esto se hace en la pestaña “File”, eligiendo la opción “Export hardware” (ver Figura 54).

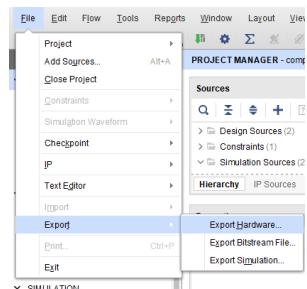


Figura 54 Exportar Hardware.

Una vez hecho esto, se abre el asistente (ver Figura 55), se tiene que incluir el bitstream (ver Figura 56) y especificar en qué ruta se guardara , al final se mostrara un resumen de las características y se da clic en Finish.

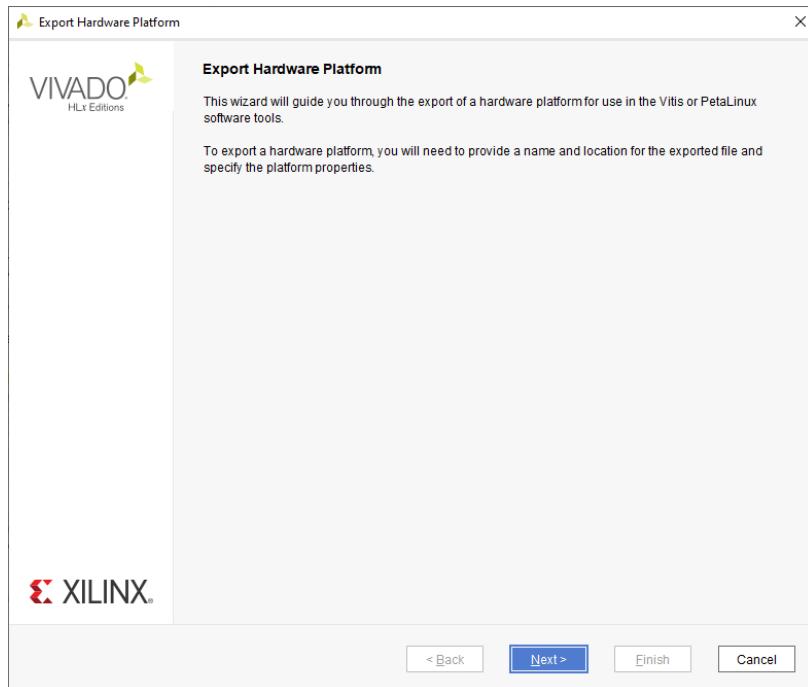


Figura 55 Asistente para exportar el hardware.

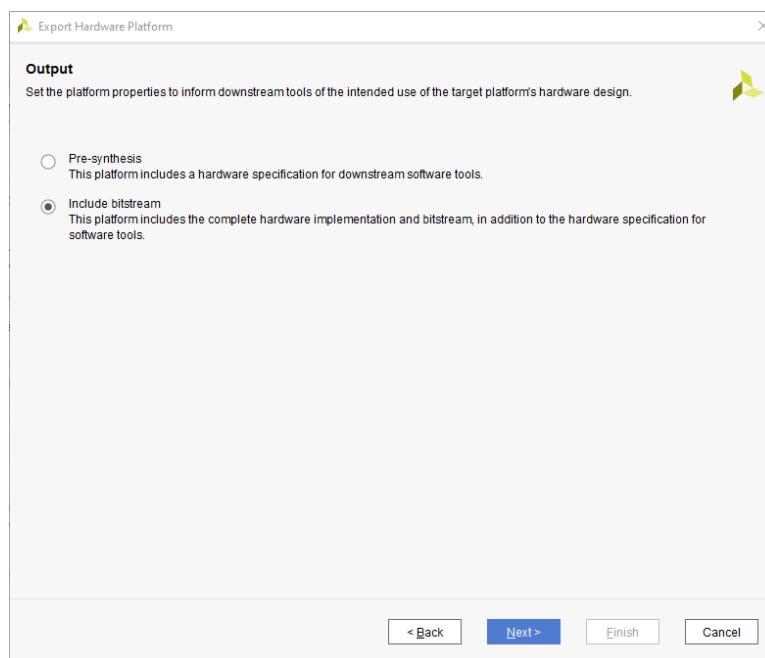


Figura 56 Incluir el bitstream.

Desde la pestaña “Tools” se puede iniciar el programa Vitis (ver Figura 56).

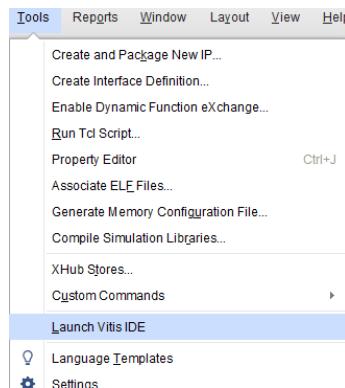


Figura 57 Iniciar Vitis IDE.

Vitis/SDK

Al abrir Vitis, se tiene que especificar el directorio de trabajo, este debe ser el mismo que se eligió al momento de exportar el hardware (ver Figura 58), una vez especificado esto, se da clic en “Launch”.

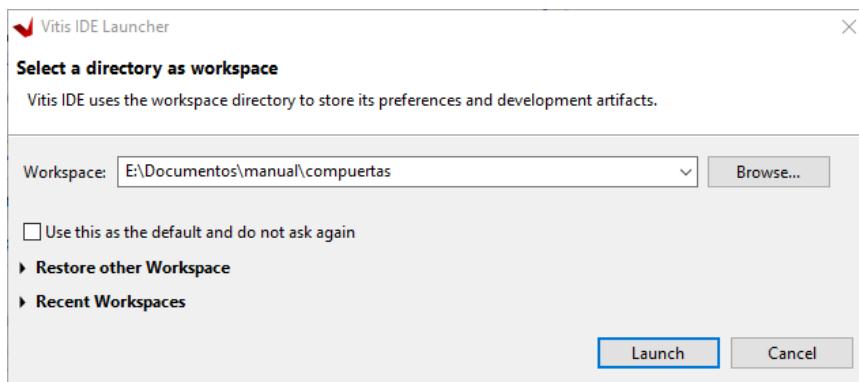


Figura 58 Selección del directorio de trabajo.

Después se tiene que crear un proyecto de aplicación, dando clic en la opción “Create Application Project” (ver Figura 59).

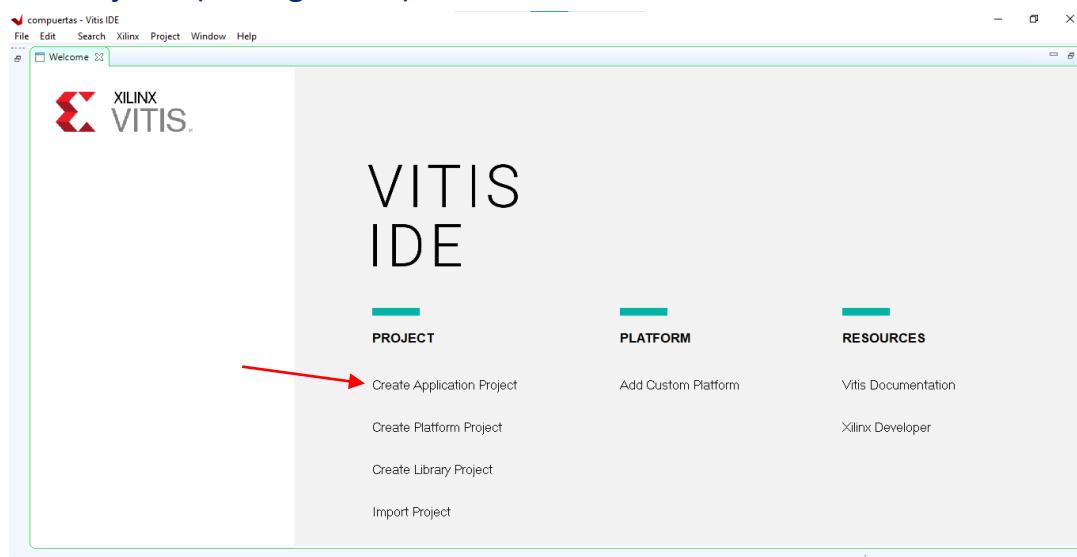


Figura 59 Crear un proyecto de aplicación.

Al hacer esto se desplegará un asistente (ver Figura 60).

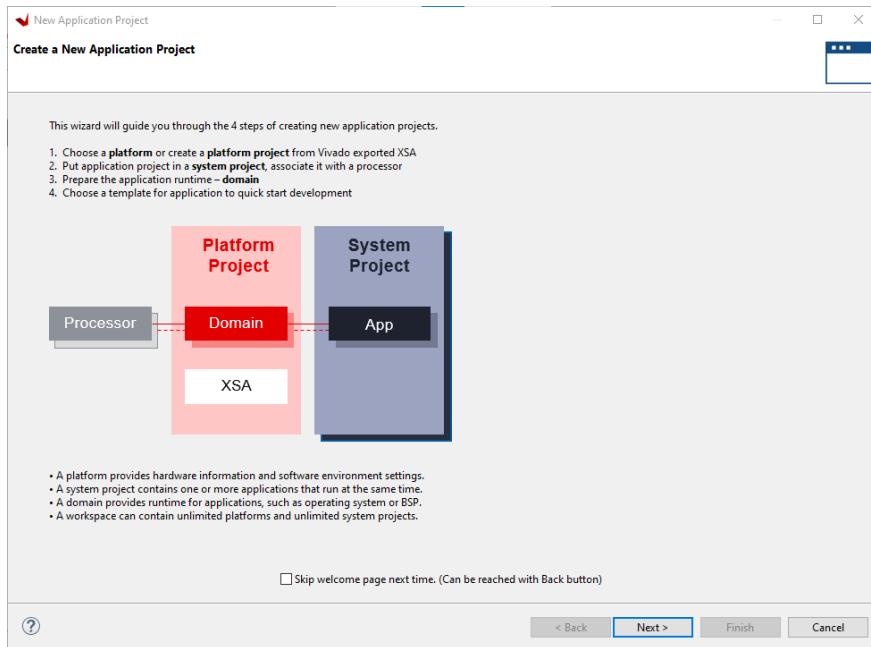


Figura 60 Asistente para crear el proyecto.

En la sección para escoger la plataforma donde se trabajara, se debe elegir la pestaña “Create a new platform form hardware (XSA)” y en “Browse..” se debe buscar el archivo XSA correspondiente al proyecto, este debe de estar en la ruta donde se trabajó en Vivado.(ver Figura 61).

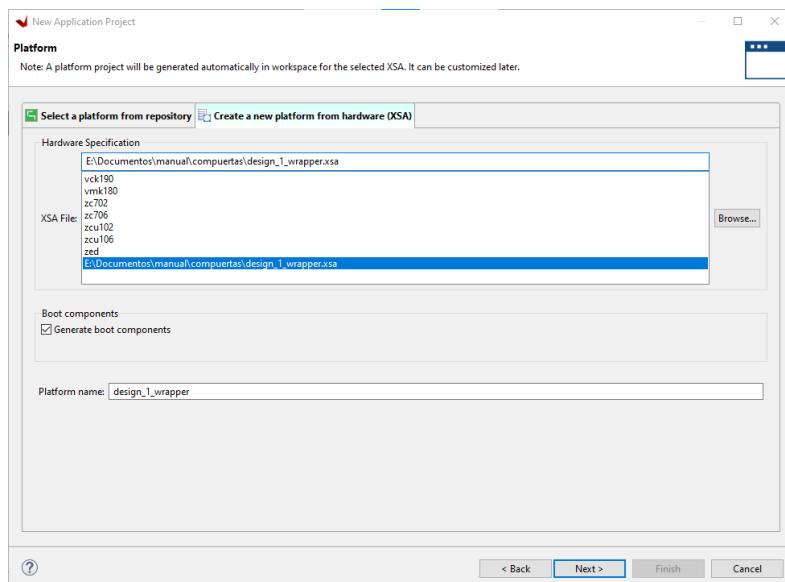


Figura 61 Selección de la plataforma.

El siguiente paso es ponerle un nombre al proyecto, una vez hecho esto se da clic en “Next”, la siguiente parte es seleccionar un dominio, en esta no se hace ningún cambio, finalmente se puede elegir un Template, en este caso se elige el “Hello World” (ver Figura 62) y se clic en “Finish”.

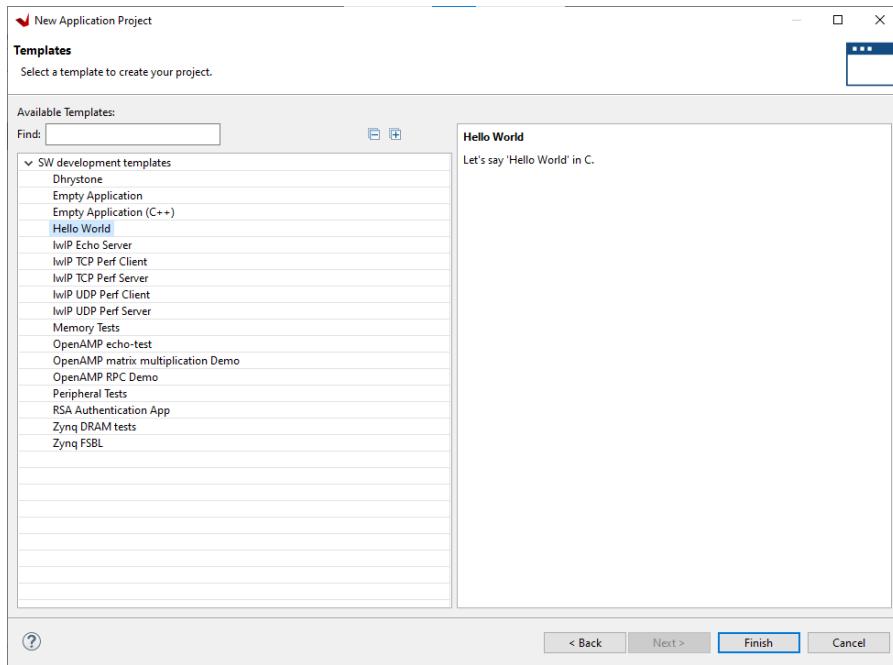


Figura 62 Elegir Hello World como template.

Una vez creado el proyecto la ventana de Vitis lucirá de manera similar a la Figura 63.

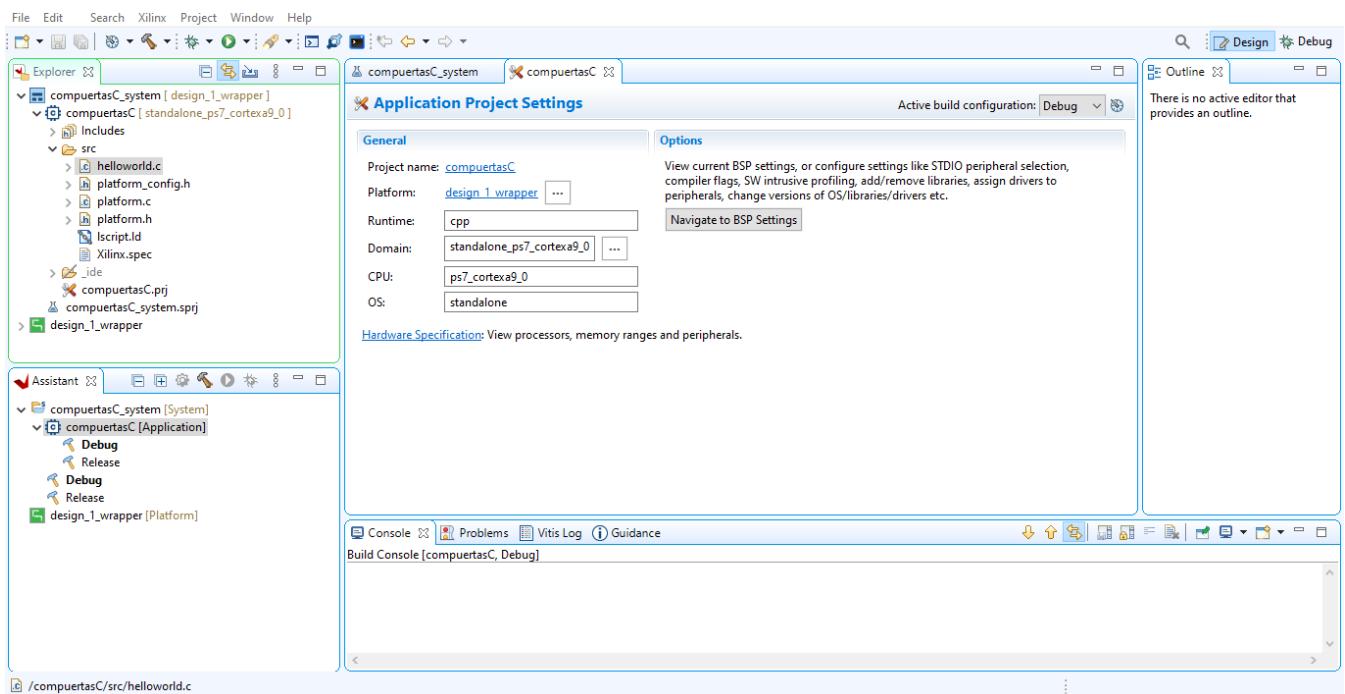


Figura 63 Proyecto creado.

Se abre el archivo “helloworld.c”, para poder visualizarlo, posteriormente se hace el “build” dando clic en el icono de martillo y estando seleccionado el proyecto, una vez finalizado se conecta la tarjeta a la computadora, después se añade una terminal serial, esto dando clic en el icono de la ventana y seleccionando “Command Shell Console” (ver Figura 64).

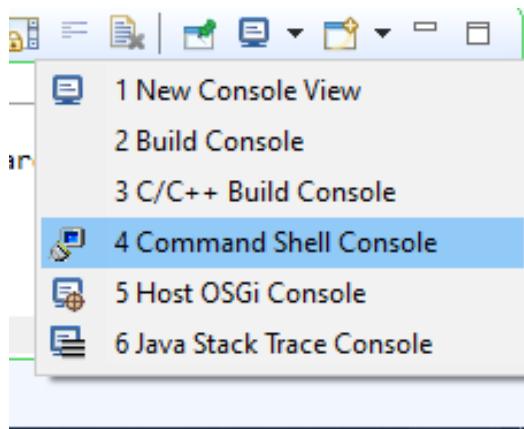


Figura 64 Command Shell Console.

Al momento de crear la conexión se debe seleccionar “Serial Port”, en “Connection name”, se crea una nueva (ver Figura 65), y se elige el encoding (ver Figura 66).

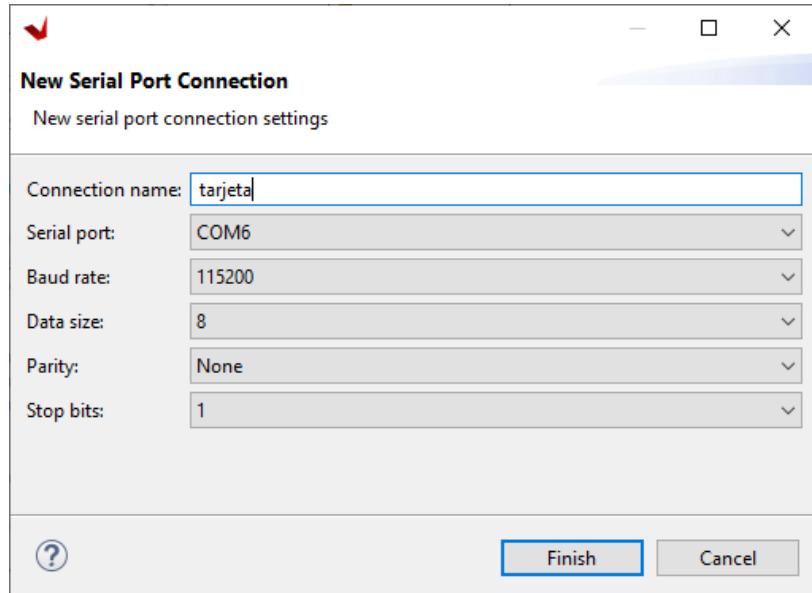


Figura 65 Nueva conexión.

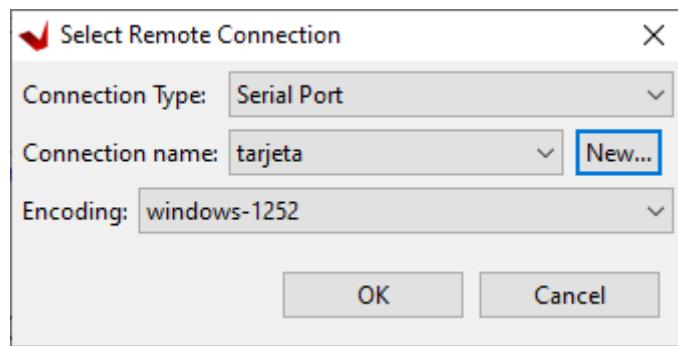


Figura 66 Seleccionar conexión remota.

Finalmente se corre la aplicación, esto teniendo seleccionado la aplicación en el explorador y dando clic en el icono de “Run” y se selecciona “Launch Hardware” (ver Figura 67).

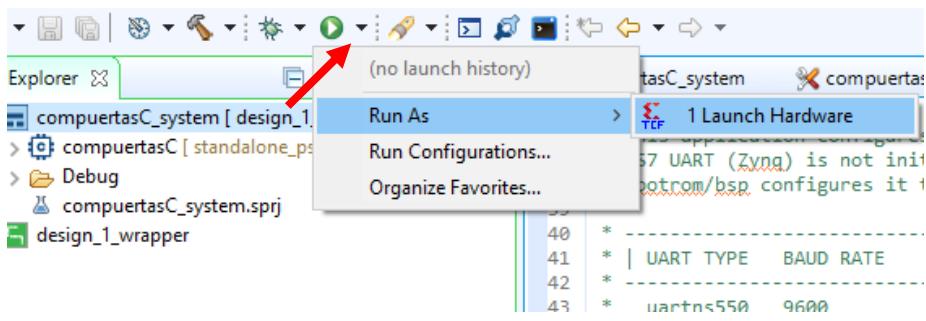


Figura 67 Launch in hardware.

Durante este proceso se programa la FPGA, al finalizar se cambia la consola desplegada, esto dando clic en el ícono del monitor (ver Figura 68) hasta encontrar la correspondiente al monitor serial (ver Figura 69).



Figura 68 Cambiar consola seleccionada.

```
tarjeta (CONNECTED)
Hello World
Successfully ran Hello World application
```

Figura 69 Comunicación del puerto serial.

Una vez probado el template, se harán las modificaciones correspondientes al código para utilizar la entrada-salida de propósito general (GPIO), el código final se muestra en el Anexo:Código C - GPIO.

Una vez modificado el archivo C, se guarda y se vuelve a hacer la build, posteriormente se ejecuta el programa.

Resultado

Una vez ejecutado la aplicación y utilizado alguna opción, la terminal serial tendrá un aspecto similar al siguiente:

```
Practica 1: ZYNQ + Modulo RTL

Elija la compuerta logica a realizar

1 AND
2 OR
3 XOR
4 NOR

Ingrese una opcion valida

5 para regresar al menu

Ingrese cualquier caracter para que se imprima el resultado

Resultado : 1

Resultado : 1
```

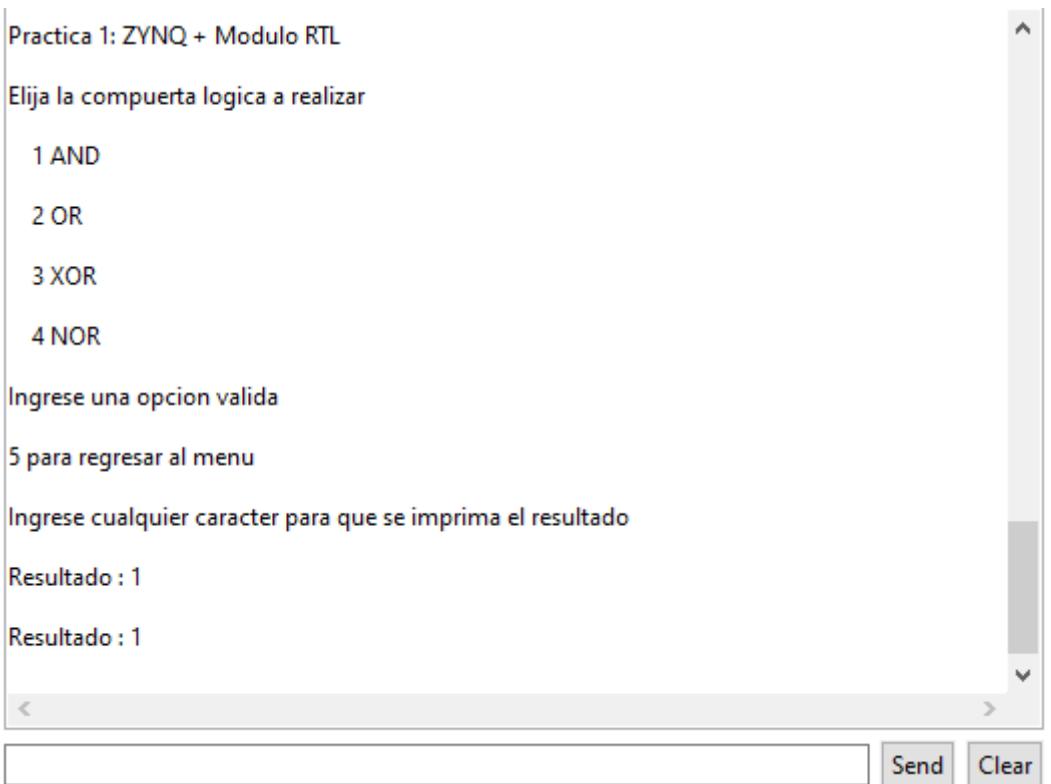


Figura 70 Resultado de la practica 1 en el monitor serial.

Práctica 2: Primer Boot

Objetivos

- Bootear PYNQ en la tarjeta de desarrollo

Desarrollo

Se requiere:

- Tarjeta PYNQ -Z2
- Computadora
- Cable Ethernet
- Cable micro USB.
- Tarjeta microSD
 - Al menos 8 GB.
 - Clase 10 o mejor.

El procedimiento es el siguiente:

Montar imagen en la microSD.

Configuracion de la tarjeta

- Red.
- Alimentacion.

Para montar la imagen es necesario descargar la versión adecuada según la versión de Xilinx.

Release version	Xilinx Tool Version
v1.4	2015.4
v2.0	2016.1
v2.1	2017.4
v2.2	2017.4
v2.3	2018.2
v2.4	2018.3
v2.5	2019.1
v2.6	2020.1

Los enlaces de descarga se encuentran en la página del fabricante. (

<https://www.tul.com.tw/ProductsPYNQ-Z2.html>)

Downloads

- PYNQ-Z2 User Manual (PDF)
- PYNQ-Z2 Boot Image
 - 1. V2.3
 - 2. V2.4
 - 3. V2.5
- PYNQ-Z2 Board File (for Pmod IP support please refere here)
- Master XDC
- Protective Acrylic Case (PDF)
- Zynq Datasheet (PDF)
- Zynq Manual (PDF)
- Schematics (PDF)

Configuración de la tarjeta.

Una vez montada la imagen en la tarjeta microSD, se realizan las conexiones de la tarjeta:

1. Ethernet.
 - Conectado a un modem
 - Conectado a una laptop.

-
- 2. Micro USB.
 - 3. Tarjeta microSD



Figura 71 Pynq Z2.

Resultado

Una vez que se encienda la PYNQ-Z2 se tiene que esperar unos momentos para que inicie el sistema, al terminar la tarjeta tendrá el siguiente aspecto.



Figura 72 Pynq Z2 después del boot

Para más información referente a la configuración y boot de la tarjeta visitar los siguientes sitios:

- https://pynq.readthedocs.io/en/latest/getting_started/pynq_z2_setup.html
- <https://youtu.be/RiFbRf6gaK4>

Práctica 3: Creación de un Overlay

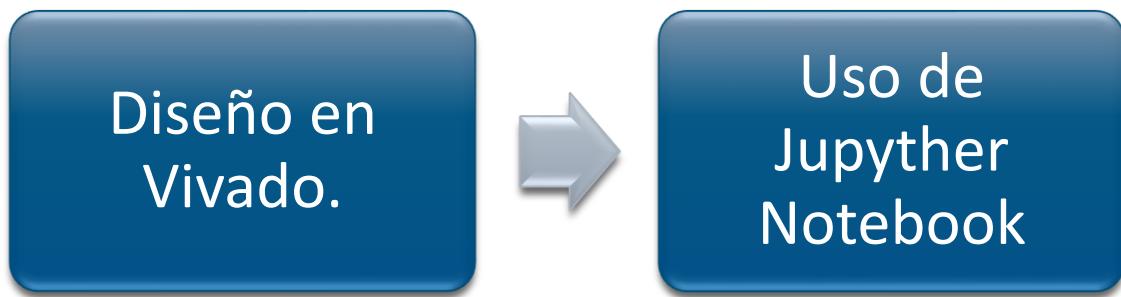
Objetivo

- Diseñar un sistema que utilice el SoC Zynq
- Generar los siguientes archivos referentes al diseño anterior:
 - Archivo Bitstream.

- Archivo. tcl
- Archivo .hwh
- Guardar los archivos en la microSD
- Elaborar aplicación y ejecutarla

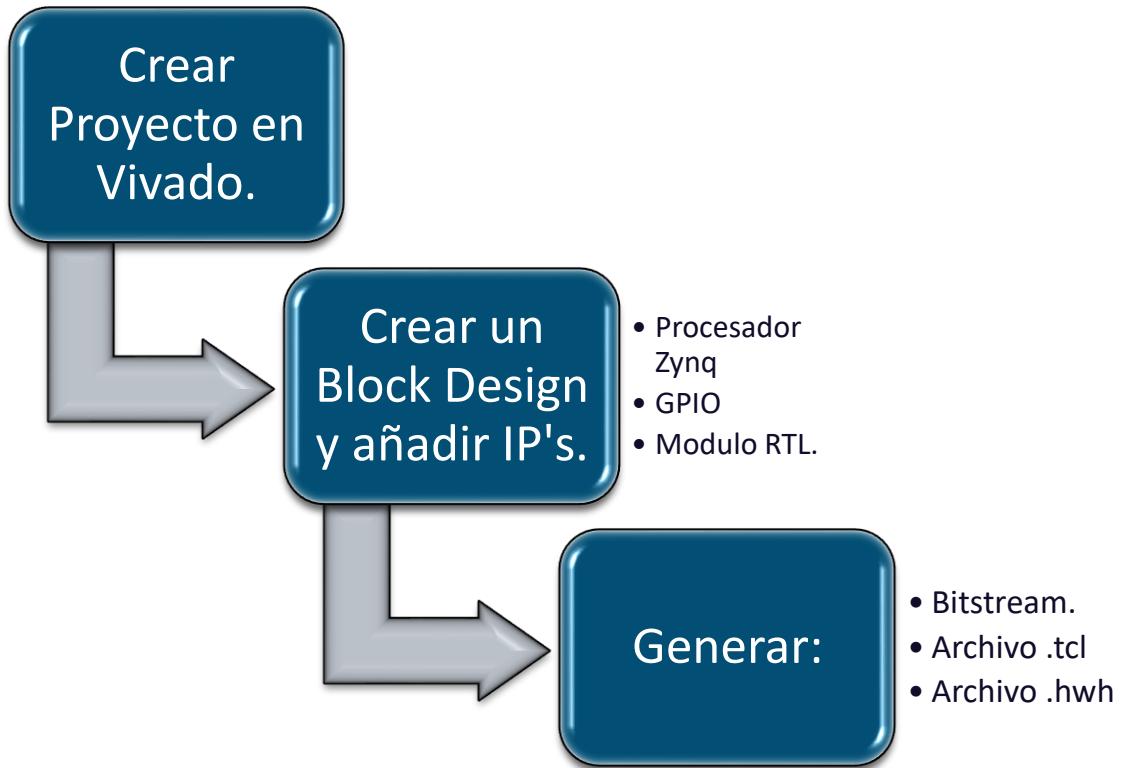
Desarrollo

Para el uso de overlays se sigue el siguiente proceso:



Diseño en Vivado

Para la sección de Vivado se tiene que realizar el siguiente proceso:



Referente a la creación del proyecto se puede revisar la sección Crear Proyecto en Vivado, en esta ocasión se utilizará el mismo diseño realizado en la sección Desarrollo Vivado.

Para exportar el Block Design es necesario que este se encuentre abierto, de otra manera no aparecerá esta opción.

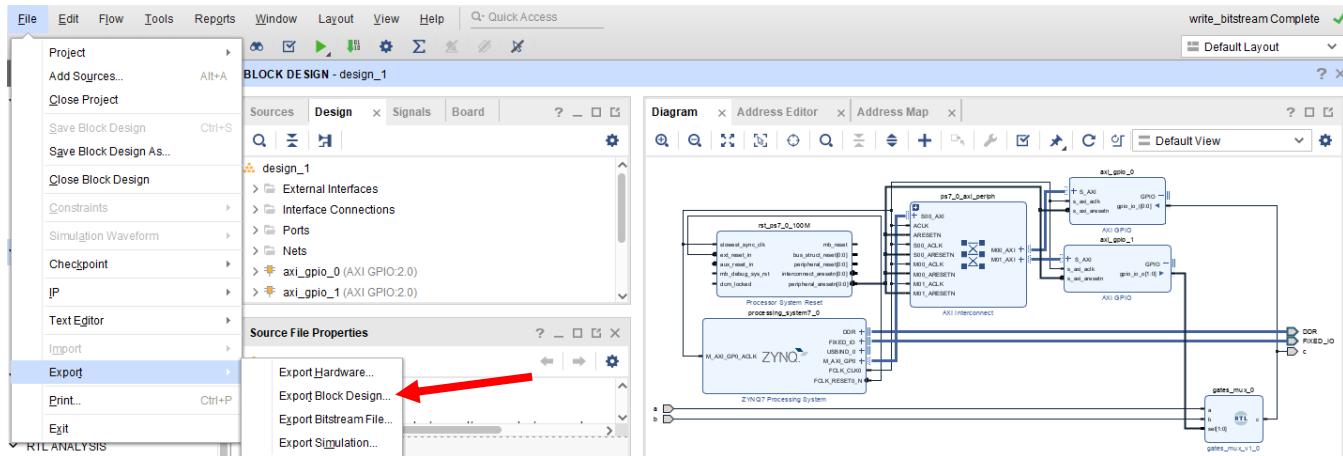


Figura 73 Exportar Block Design.

El archivo “.tcl” se genera al exportar el Block Design y el archivo “.hwh” se crea al generar el bitstream, es recomendable guardar estos 3 archivos en una carpeta individual.

Uso de Jupyter Notebook

Para esa etapa se seguirá el siguiente proceso:



Para el proceso anterior se debe tener ya conectada a la tarjeta.

Guardar archivos en la tarjeta

Desde el explorador de archivos se accede a la siguiente ruta:
Red>Nombre_Tarjeta>xilinx.

El nombre de usuario y contraseña es xilinx.

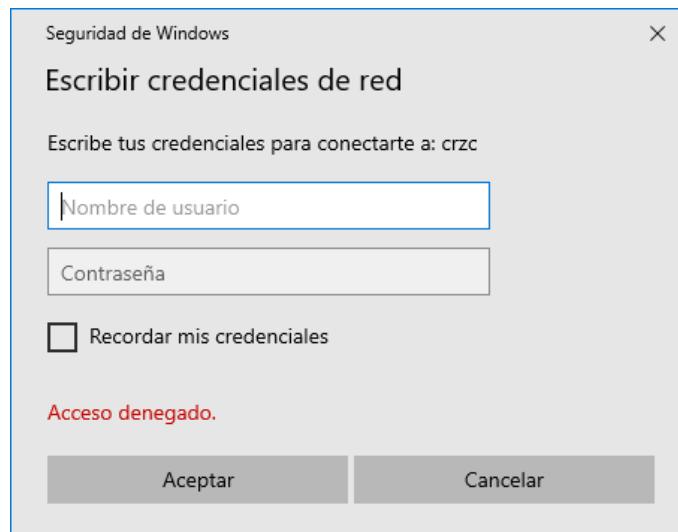


Figura 74 Validación de las credenciales en Windows.

En la ubicación se encuentran los siguientes documentos:

Nombre	Fecha de modificación	Tipo	Tamaño
jupyter_notebooks	16/02/2021 07:58 p. m.	Carpeta de archivos	
pynq	30/09/2019 07:04 a. m.	Carpeta de archivos	
REVISION	03/10/2019 12:35 p. m.	Archivo	1 KB

Figura 75 Documentos dentro de la tarjeta.

Dentro de la carpeta jupyter_notebooks se crea una carpeta en donde deben estar ubicados los 4 archivos generados anteriormente, se recomienda colocarle un nombre para que los identifique:

Red > crzc > xilinx > jupyter_notebooks > compuertas

Nombre	Fecha de modificación	Tipo	Tamaño
compuertas.bit	22/01/2021 04:47 p. m.	Archivo BIT	3,951 KB
compuertas.hwh	22/01/2021 04:43 p. m.	Archivo HWH	182 KB
compuertas.tcl	24/01/2021 07:40 p. m.	Archivo TCL	47 KB

Una vez hecho esto en un navegador se va la siguiente url: http://Nombre_tarjeta:9090/, en caso de que pida iniciar sesión, la contraseña es xilinx.

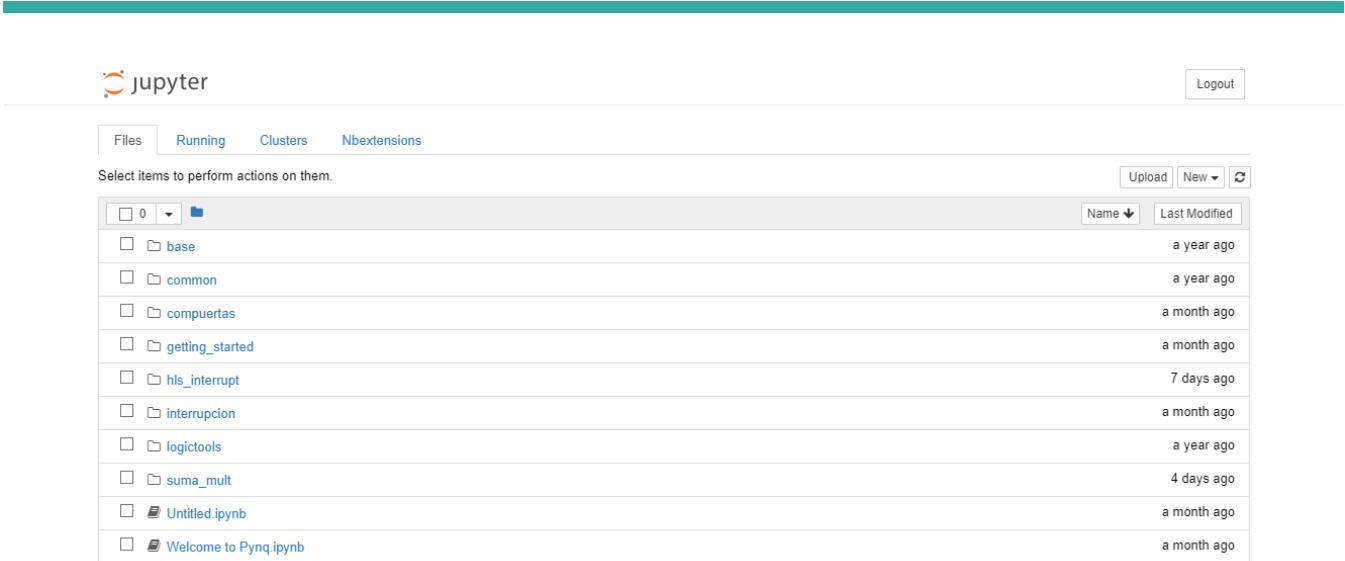


Figura 76 Jupyter notebook.

Una vez abierto la carpeta que se creó anteriormente, se crea una notebook de Python 3.

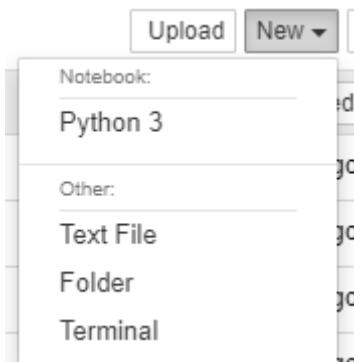


Figura 77 Crear un nuevo archivo.

En general para cualquier proyecto con overlays se sigue el siguiente proceso:



Importar Overlay.

Se realiza con las siguientes dos líneas, esto tomando en cuenta que la libreta y el archivo se encuentren en la misma carpeta, de lo contrario se tiene que especificar la ruta:

```
In [1]: from pynq import Overlay
ol = Overlay("./compuertas.bit")
```

Un paso opcional es comprobar que se haya cargado correctamente, esto se hace con la siguiente línea:

ol?

Después de ejecutar ambas celdas aparece una ventana con la información del overlay importado y las características de cada IP o componente que contiene:

```
Type:          Overlay
String form:  <pynq.overlay.Overlay object at 0xb54724f0>
File:         /usr/local/lib/python3.6/dist-packages/pynq/overlay.py
Docstring:
Default documentation for overlay ./compuertas.bit. The following
attributes are available on this overlay:

IP Blocks
-----
axi_gpio_0      : pynq.lib.axigpio.AxiGPIO
axi_gpio_1      : pynq.lib.axigpio.AxiGPIO

Hierarchies
-----
None

Interrupts
-----
None

GPIO Outputs
-----
None

Memories
-----
```

Definir variables.

En esta etapa se declaran las variables referentes a las IP's del Overlay, esto para que posteriormente sea más fácil utilizarlo.

```
seleccion=ol.axi_gpio_0
salida=ol.axi_gpio_1
```

Programar.

Para realizar una interfaz amigable al usuario se realizó el siguiente código:

```

import csv
import sys

def main():
    menu()

def menu():
    print("*****Segunda Practica ZYNQ + PYTHON *****")
    print()

    choice = input("""
        A: AND %d
        B: OR
        C: XOR
        D: NOR

        Ingrese selección: """)

    if choice == "A" or choice == "a":
        and_op()
        print("Resultado ", salida.read(0x00))
    elif choice == "B" or choice == "b":
        or_op()
        print("Resultado ", salida.read(0x00))
    elif choice == "C" or choice == "c":
        xor_op()
        print("Resultado ", salida.read(0x00))
    elif choice == "D" or choice == "d":
        nor_op()
        print("Resultado ", salida.read(0x00))
    else:
        print("Ingrese una opción valida")
        print("Intente de nuevo")
        menu()

def and_op():
    pass
    print("Realizando AND")
    seleccion.write(0x0,0)

def or_op():
    pass
    print("Realizando OR")
    seleccion.write(0x0,1)

def xor_op():
    pass
    print("Realizando XOR")
    seleccion.write(0x0,2)

def nor_op():
    pass
    print("Realizando NOR")
    seleccion.write(0x0,3)

while 1:
    main()

```

Figura 78 Código propuesto.

Las dos instrucciones importantes son:

- salida.read
 - Se tiene que especificar la dirección.
- Selección.write
 - Se tiene que especificar la dirección.
 - Se tiene que especificar el valor a escribir.

Resultado

Una vez ejecutado las celdas anteriores se obtiene lo siguiente:

```
*****Segunda Practica ZYNQ + PYTHON *****
```

A: AND %d
B: OR
C: XOR
D: NOR

Ingrese selección: A

Realizando AND

Resultado 1

```
*****Segunda Practica ZYNQ + PYTHON *****
```

A: AND %d
B: OR
C: XOR
D: NOR

Ingrese selección:

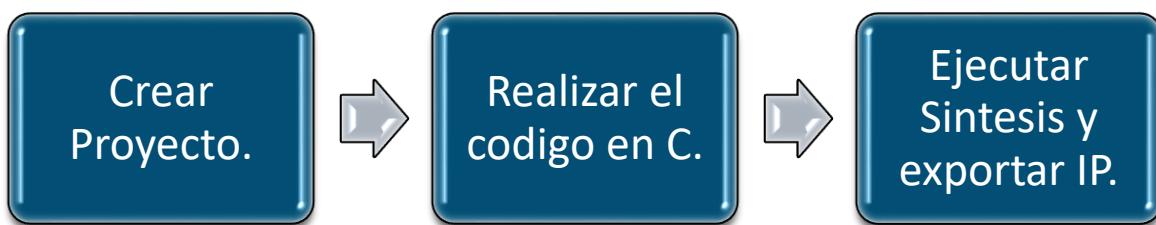
Práctica 4: Crear IP con Vitis HLS

Objetivo

- Crear una función en C que realiza la suma de 2 números
- Crear una IP de la función anterior

Desarrollo

Para utilizar Vitis HLS, se seguirá el siguiente procedimiento:



Crear Proyecto

Una vez abierto Vitis HLS, se muestra la página de inicio, donde se da clic en la opción “Create Project” como se muestra en la Figura 79.

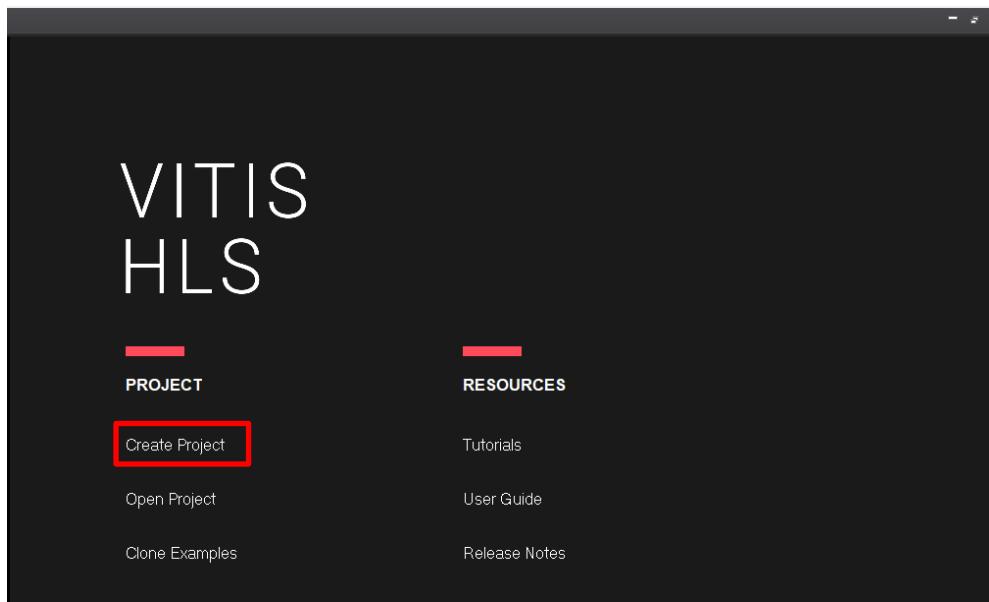


Figura 79 Pagina de Bienvenida de Vitis HLS.

Después se desplegará el asistente, el primer paso, es elegir un nombre para el proyecto y su ubicación (ver Figura 80). Se recomienda que la ruta no sea muy larga para evitar errores.

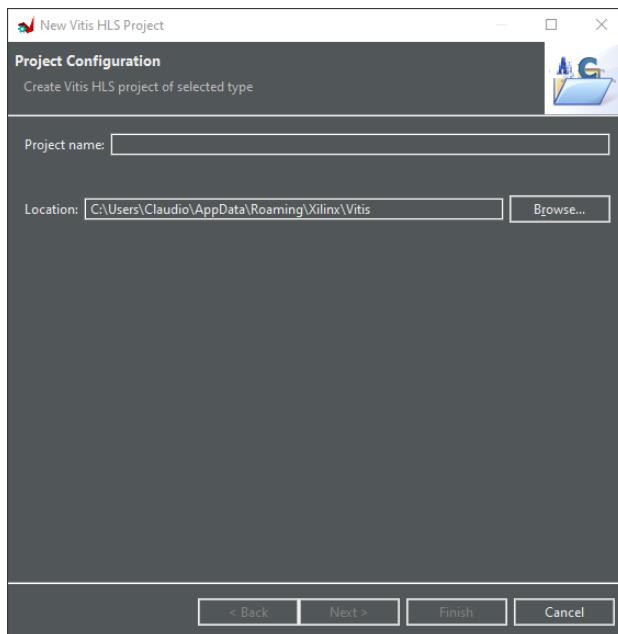


Figura 80 Vitis HLS, nombre y ubicación del archivo.

Lo siguiente es agregar los archivos de diseño, se especifica a “resta” como función top y se añade el archivo de diseño usando la opción “Add Files”, se desplegará otra venta, donde se elige la ubicación y nombre del archivo, este debe ser .cpp. (ver Figura 81)

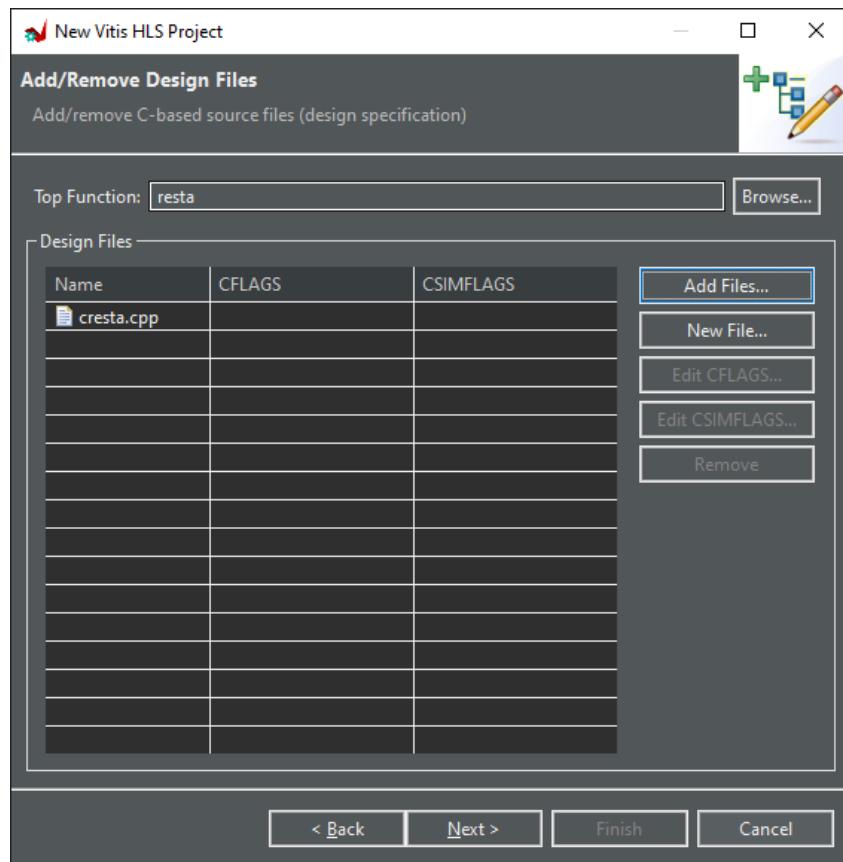


Figura 81 Agregar-remover archivos de diseño.

El siguiente paso es agregar algún archivo Testbench, sin embargo, en este caso no se hará, finalmente se termina con la configuración de la solución, donde se elige un nombre y la tarjeta (ver Figura 82).

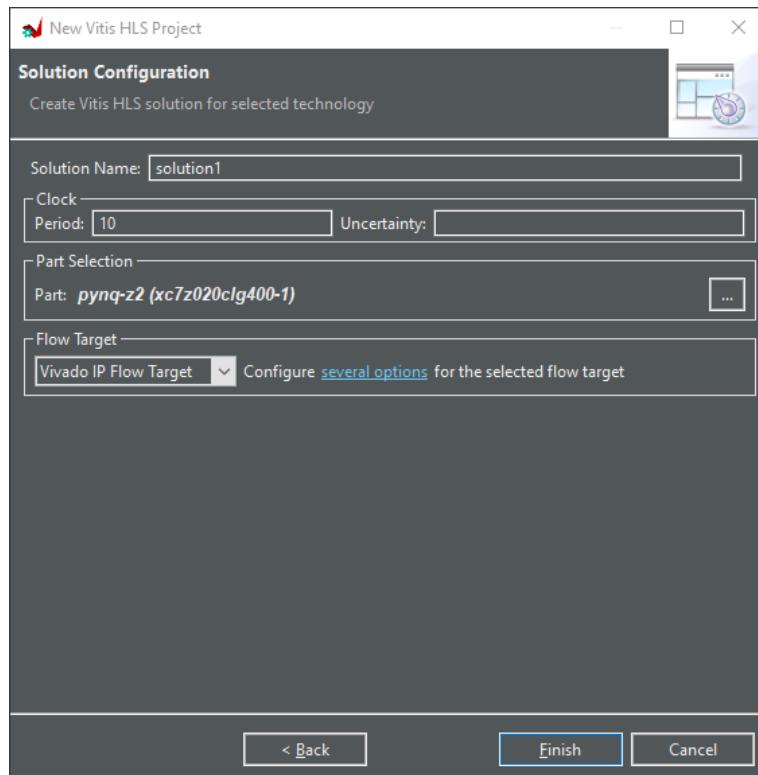


Figura 82 Configuración de la solución.

Realizar Código en C

El código para implementar es una suma:

```
void suma (float *a, float *b, float *c)
{
#pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
    *c += *a + *b;
}
```

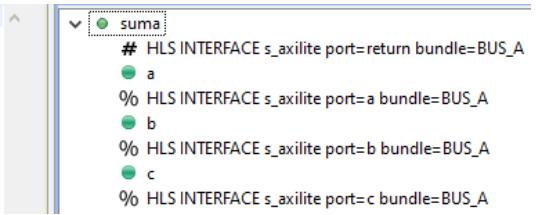


Figura 83 Código en C utilizado en Vitis HLS.

Ejecutar Síntesis y exportar IP

Una vez guardado las modificaciones se ejecuta la síntesis.

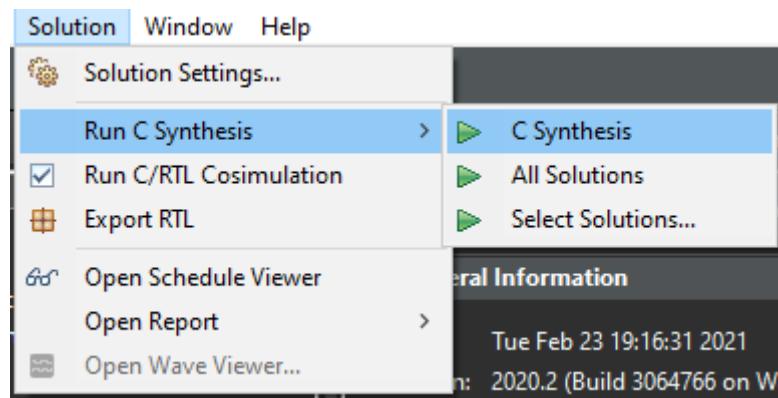


Figura 84 Ejecutar C síntesis.

Una vez terminado el proceso se exporta la IP.

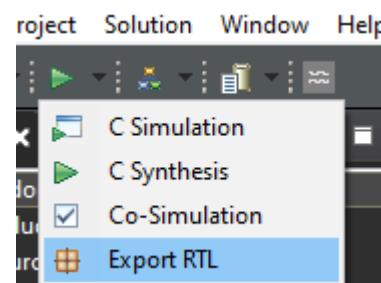


Figura 85 Exportar RTL.

Se recomienda que la ubicación de los archivos generados no sea una ruta muy larga ya que podría ocasionar problemas, el lenguaje se escoge a VHDL sin embargo también se puede optar por Verilog.

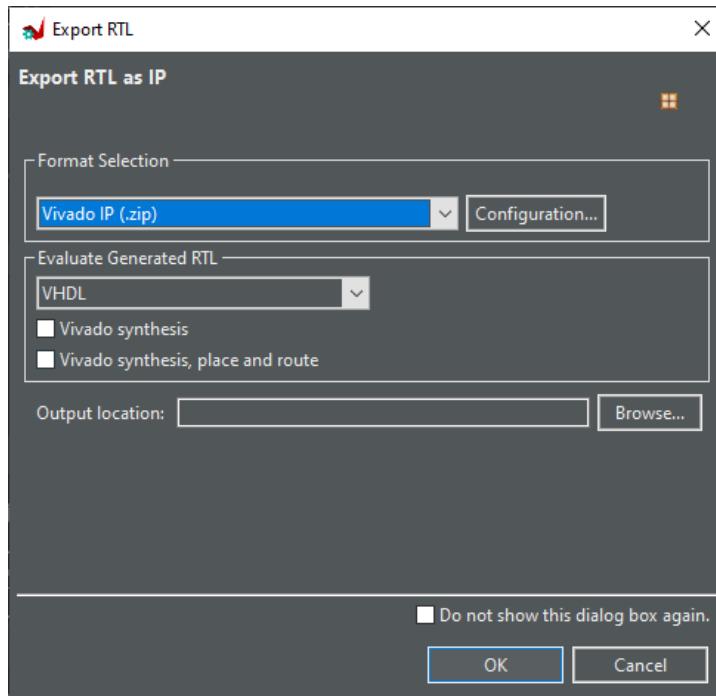


Figura 86 Asistente para exportar la IP.

Resultado

Una vez exportado la IP, la carpeta o archivo comprimido se encontrará en la ubicación del proyecto, como se muestra en la siguiente figura:

Nombre	Fecha de modificación	Tipo	Tamaño
ip	14/04/2021 11:11 a. m.	Carpeta de archivos	
misc	14/04/2021 10:59 a. m.	Carpeta de archivos	
verilog	14/04/2021 11:11 a. m.	Carpeta de archivos	
vhdl	14/04/2021 11:11 a. m.	Carpeta de archivos	

Figura 87 Carpeta de la IP una vez que se ha exportado.

Práctica 5: Overlay personalizado con IP's

Objetivo

- Crear un diseño donde se comunique el SoC Zynq con una IP creada en Vitis HLS
- Crear una aplicación en SDK/Vitis
- Crear una aplicación en Python

Desarrollo

El proceso que se realiza es el mismo que el descrito en la sección Práctica 3: Creación de un Overlay únicamente en la parte correspondiente a Vivado se tiene que agregar la IP.

Vivado.

Una vez creado el proyecto, se añade la IP, se puede realizar desde Settings.

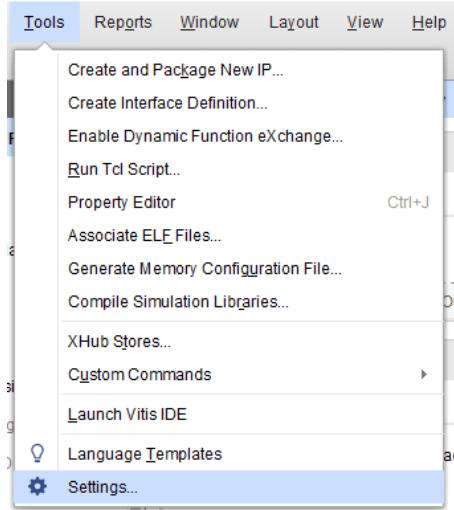


Figura 88 Ajustes del proyecto.

En la sección IP>Repository, se puede añadir todas las IPs que se quieran al diseño.

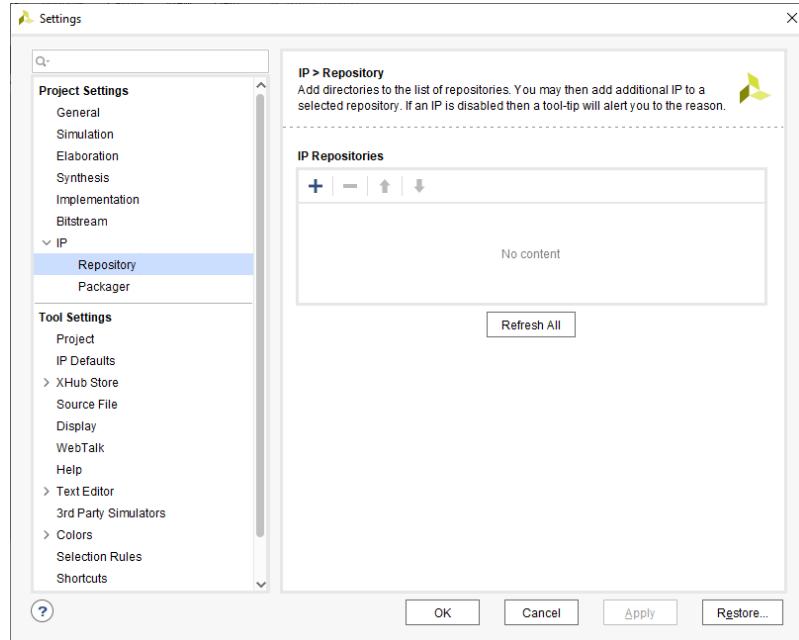


Figura 89 Repositorio de IP's.

Una vez agregado se aplican los cambios y se da clic en OK.

Nota: Si al momento de exportar la IP únicamente se generó un archivo .zip es necesario descomprimir este archivo para que lo reconozca Vivado.

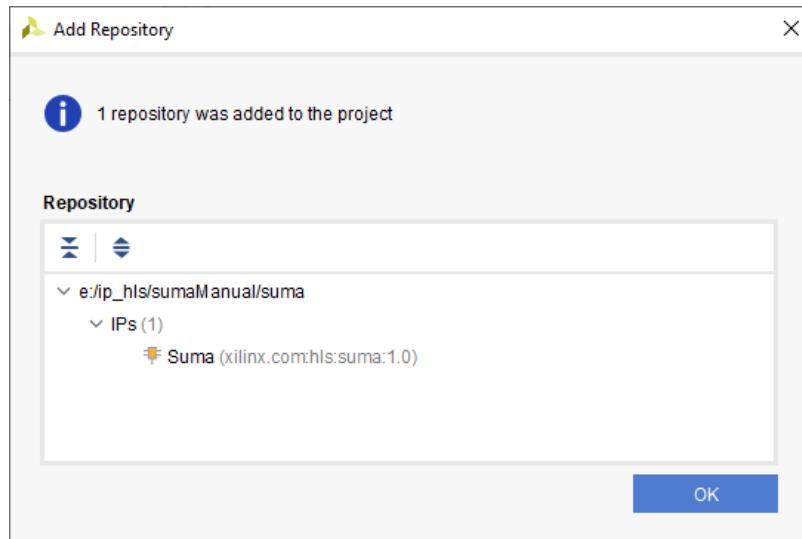


Figura 90 Repositorio añadido.

Para el diseño se utilizarán únicamente las siguientes IPs:

- Procesador Zynq
- Suma
- Control de Interrupciones.

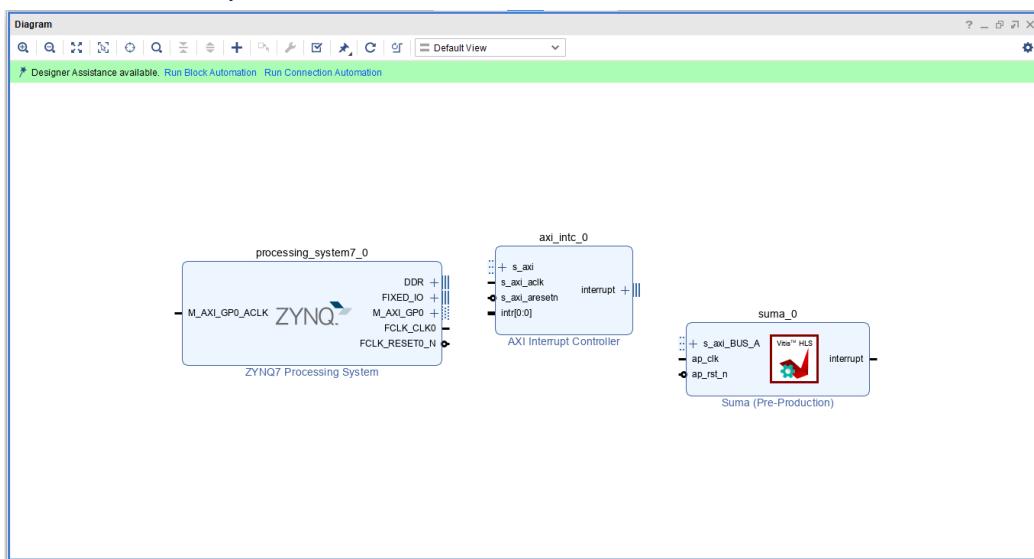


Figura 91 IPs necesarias en el diseño.

Se utiliza el Run Block Automation con las conexiones predeterminadas.

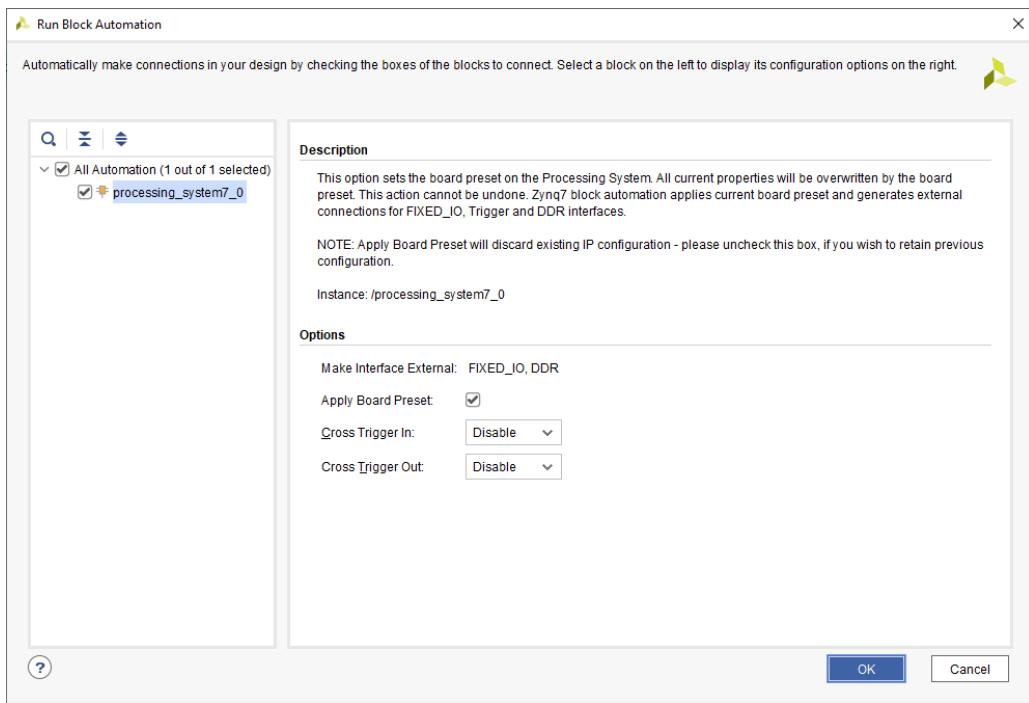


Figura 92 Configuración para el Run Block Automation.

Después se utiliza Run Connection Automation, habilitando las siguientes conexiones:

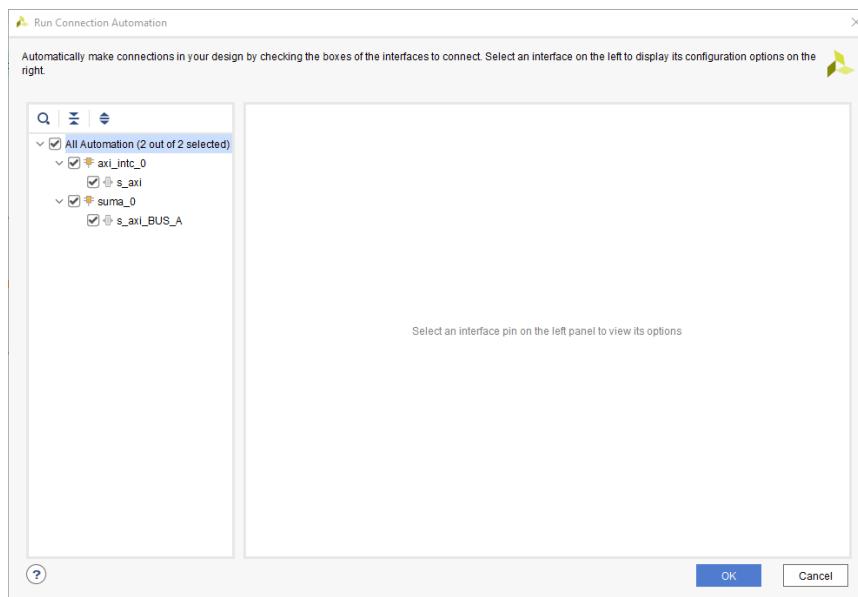


Figura 93 Configuración para Run Connection Automation.

Una vez terminado, se tiene que habilitar las interrupciones del procesador, esto se hace dando doble clic sobre el bloque del procesador y en la sección Interrupts se habilitan las interrupciones de fábrica.

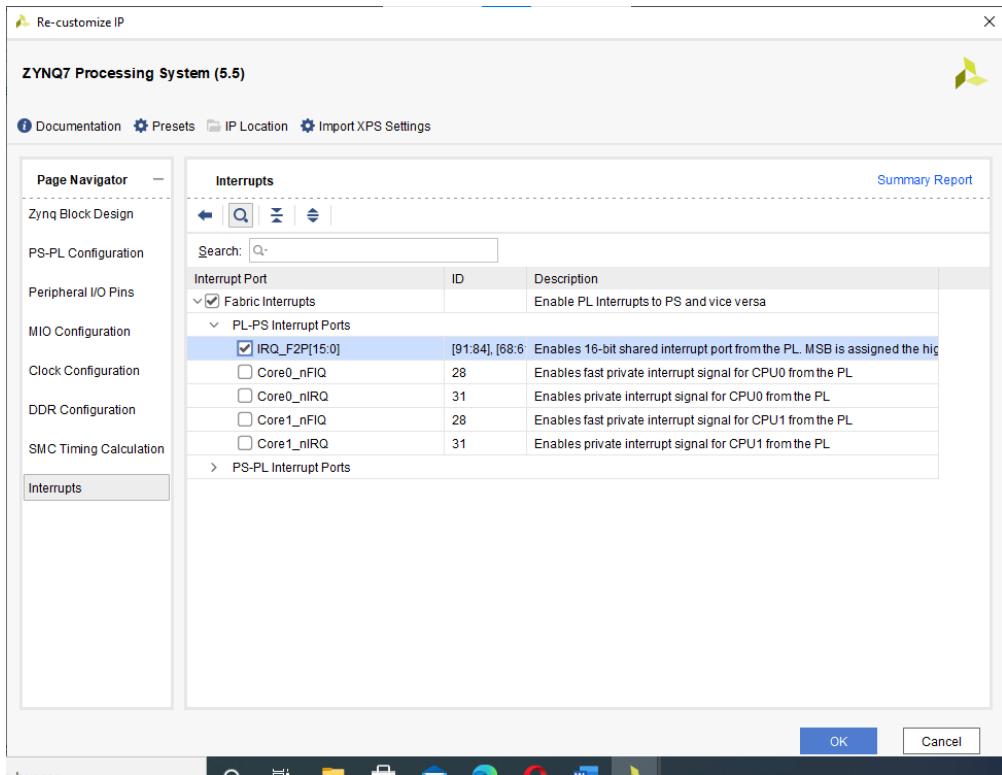


Figura 94 Interrupciones disponibles.

Después de habilitar las interrupciones podremos ver una entrada nueva en el procesador:

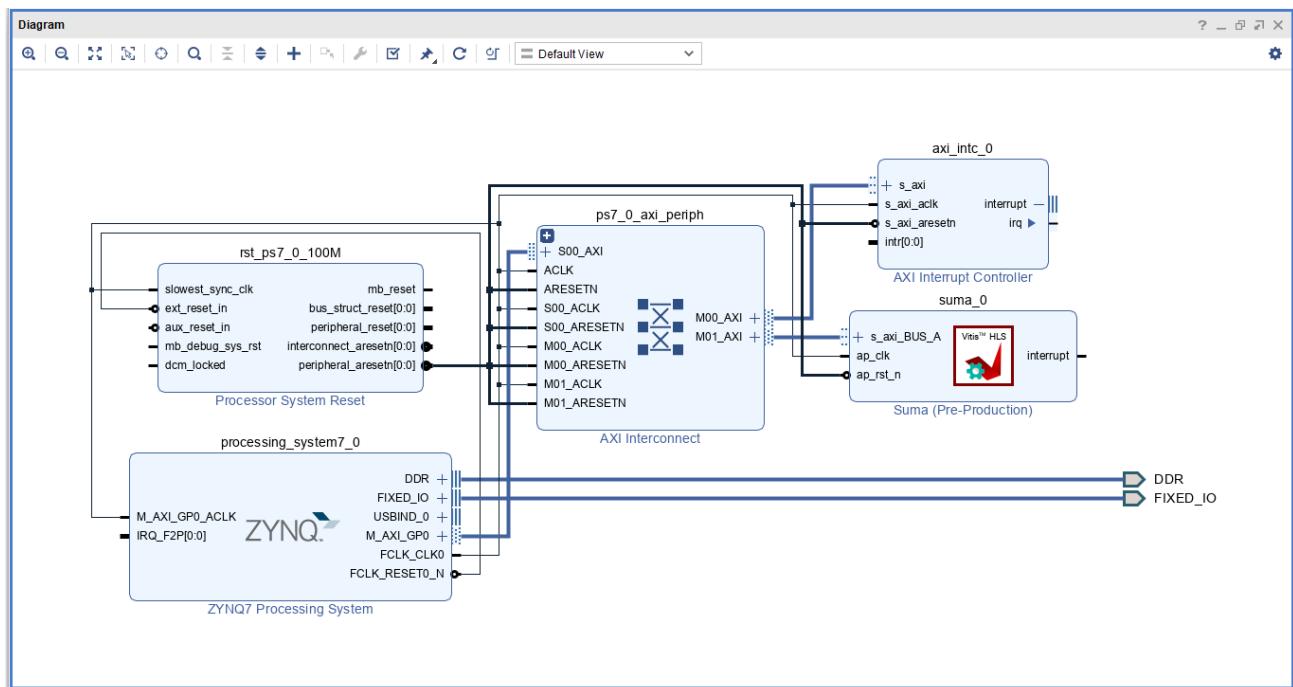


Figura 95 Diagrama con las conexiones.

Por lo que se realizan las conexiones resaltadas en color naranja:

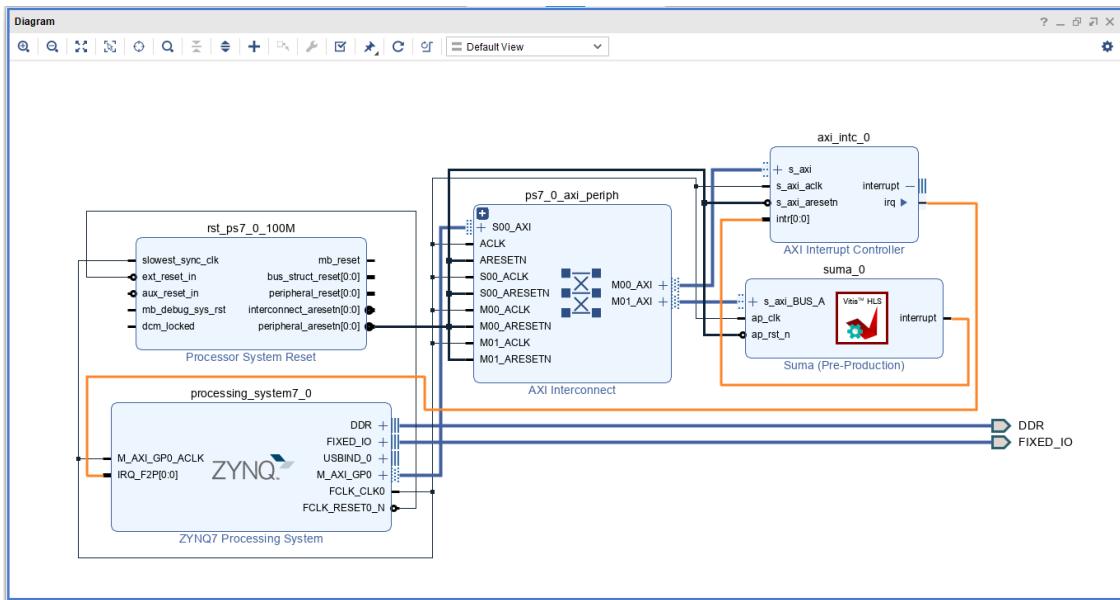


Figura 96 Conexiones necesarias para la interrupción.

Finalmente se tienen que generar los archivos mencionados en la Diseño en Vivado Jupyter

Para esta parte se tiene que realizar el mismo proceso que en la sección Uso de Jupyter Notebook, el único cambio es referente a la programación en Python.

```
In [1]: from pynq import Overlay
overlay = Overlay("./practica5.bit")

In [2]: overlay?

Type:          Overlay
String form:  <pynq.overlay.Overlay object at 0xb542df30>
File:          /usr/local/lib/python3.6/dist-packages/pynq/overlay.py
Docstring:
Default documentation for overlay ./practica5.bit. The following
attributes are available on this overlay:

IP Blocks
-----
axi_intc_0      : pynq.overlay.DefaultIP
suma_0          : pynq.overlay.DefaultIP

Hierarchies
-----
```

Figura 97 Celdas y resultado obtenido.

Después se asigna un nombre más amigable a la IP de la suma y además se visualizan las características referentes a interrupciones de la IP.

```
In [3]: suma=overlay.suma_0
suma._interrupts

Out[3]: {'interrupt': {'controller': 'axi_intc_0',
  'fullpath': 'suma_0/interrupt',
  'index': 0}}
```

Figura 98 Características de la IP respecto a la interrupción.

También se le asigna el atributo de la interrupción a una variable y se muestra los registros correspondientes a la IP.

```
In [5]: interrupt = suma.interrupt
suma.register_map

Out[5]: RegisterMap {
    CTRL = Register(AP_START=0, AP_DONE=0, AP_IDLE=1, AP_READY=0, RESERVED_1=0, AUTO_RESTART=0, RESERVED_2=0),
    GIER = Register(Enable=0, RESERVED=0),
    IP_IER = Register(CHAN0_INT_EN=0, CHAN1_INT_EN=0, RESERVED=0),
    IP_ISR = Register(CHAN0_INT_ST=0, CHAN1_INT_ST=0, RESERVED=0),
    a = Register(a=0),
    b = Register(b=0),
    c_i = Register(c_i=0),
    c_o = Register(c_o=0),
    c_o_ctrl = Register(c_o_ap_vld=0, RESERVED=0)
}
```

Figura 99 Mapa de los registros de la IP.

Adicionalmente se pueden tomar en cuenta los drivers generados por Vitis HLS, estos se encuentran en la carpeta drivers>Nombre_IP>src, en este caso utilizaremos el archivo “xsuma_hw.h”.

equipo > Almacenamiento (E:) > ip_hls > sumaManual > suma > drivers > suma_v1_0 > src				
Nombre	Fecha de modificación	Tipo	Tamaño	
Makefile	23/02/2021 07:28 p. m.	Archivo	2 KB	
xsuma.c	23/02/2021 07:28 p. m.	Archivo C	7 KB	
xsuma.h	23/02/2021 07:28 p. m.	Archivo H	4 KB	
xsuma_hw.h	23/02/2021 07:28 p. m.	Archivo H	3 KB	
xsuma_linux.c	23/02/2021 07:28 p. m.	Archivo C	5 KB	
xsuma_sinit.c	23/02/2021 07:28 p. m.	Archivo C	2 KB	

Figura 100 Drivers generados por Vitis HLS.

En el cual se encuentran las diferentes señales de control y sus direcciones.

```

' BUS_A
' 0x00 : Control signals
'     bit 0 - ap_start (Read/Write/SC)
'     bit 1 - ap_done (Read/COR)
'     bit 2 - ap_idle (Read)
'     bit 3 - ap_ready (Read)
'     bit 7 - auto_restart (Read/Write)
'     others - reserved
' 0x04 : Global Interrupt Enable Register
'     bit 0 - Global Interrupt Enable (Read/Write)
'     others - reserved
' 0x08 : IP Interrupt Enable Register (Read/Write)
'     bit 0 - enable ap_done interrupt (Read/Write)
'     others - reserved
' 0x0c : IP Interrupt Status Register (Read/TOW)
'     bit 0 - ap_done (COR/TOW)
'     others - reserved
' 0x10 : Data signal of a
'     bit 7~0 - a[7:0] (Read/Write)
'     others - reserved
' 0x14 : reserved
' 0x18 : Data signal of b
'     bit 7~0 - b[7:0] (Read/Write)
'     others - reserved
' 0x1c : reserved
' 0x20 : Data signal of c_i
'     bit 7~0 - c_i[7:0] (Read/Write)
'     others - reserved
' 0x24 : reserved
' 0x28 : Data signal of c_o
'     bit 7~0 - c_o[7:0] (Read)
'     others - reserved
' 0x2c : Control signal of c_o
'     bit 0 - c_o_ap_vld (Read/COR)
'     others - reserved
' (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)

```

Figura 101 Driver de la IP.

Para utilizar las interrupciones se utiliza la librería `asyncio`, esto para utilizar las co-rutinas y la librería `time` para obtener el tiempo de ejecución, además se crearon dos funciones: `hex_to_float` y `float_to_hex`, esto para hacer la conversión del tipo de dato para enviarlo a la IP y para posteriormente tomar el resultado. También se creó una función menú, donde se realiza la adquisición de los valores A y B de la suma.

```

import asyncio
import time
import struct
import csv
import sys
from ctypes import *

def main():
    menu()

def menu():
    print("*****Práctica 5 ZYNQ + PYTHON *****")
    print()
    print("          C = A + B      \n")
    valorA = float(input(" Ingrese el Valor de A"))
    valorB = float(input(" Ingrese el Valor de B"))
    A=float_to_hex(valorA)
    B=float_to_hex(valorB)
    loop = asyncio.get_event_loop()
    loop.run_until_complete(sumaonIP(A,B))

def hex_to_float(s):
    i = int(s, 16)
    cp = pointer(c_int(i))
    fp = cast(cp, POINTER(c_float))
    return fp.contents.value

def float_to_hex(value):
    return struct.pack('f', value)

async def sumaonIP(A,B):
    inicio=time.time()
    suma.write(0x004,1)
    suma.write(0x008,1)
    suma.write(0x010,A)
    suma.write(0x018,B)
    suma.write(0x008,1)
    await interrupcion.wait()
    final=time.time()
    a=suma.read(0x028)
    b = hex_to_float(hex(a))
    totals=final-inicio
    print(f"la suma IP tardó: {totals} segundos")
    print(f"la suma es {b}")
    suma.write(0x000,128)

while 1:
    main()

```

Figura 102 Código propuesto.

En la sección de la IP es necesario habilitar las interrupciones, esto de acuerdo con la información de los drivers.

SDK

Una vez exportado el hardware, se crea una aplicación, utilizando el lenguaje C, se recomienda utilizar como base el template de “Hello World”. El código propuesto es el siguiente:

```

#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include <xsuma.h>

XSuma doSum;
XSuma_Config *doSumCfg;

void iniciar_Suma(){
    int status=0;
    doSumCfg = XSuma_LookupConfig(XPAR_SUMA_0_DEVICE_ID);
    if (doSumCfg){
        status = XSuma_CfgInitialize(&doSum,doSumCfg);
        if (status != XST_SUCCESS){

```

```

        printf("Ha ocurrido un error al iniciar la IP :( \n");
    }
}
unsigned int float_a_u32(float val){
    unsigned int resultado;
    union float_bytes{
        float v;unsigned char bytes[4];
    }data;
    data.v = val;
    resultado = (data.bytes[3] << 24) + (data.bytes[2] << 16) + (data.bytes[1] << 8) +
(data.bytes[0]);
    return resultado;
}
float u32_a_float(unsigned int val){
    union{
        float val_float;
        unsigned char bytes[4];
    }data;
    data.bytes[3] = (val >> (8*3)) & 0xff;
    data.bytes[2] = (val >> (8*2)) & 0xff;
    data.bytes[1] = (val >> (8*1)) & 0xff;
    data.bytes[0] = (val >> (8*0)) & 0xff;
    return data.val_float;
}
int main()
{
    float a;
    float b;
    float c = 0;
    init_platform();
    iniciar_Suma();
    while(1){
        printf("-----Programa para utilizar la IP - Suma \n");
        printf("          C = A + B \n");
        printf("Donde C, A y B son datos tipo float \n");
        printf("Ingrese valor de A : \n");
        scanf("%f",&a);
        printf("Ingrese valor de B : \n");
        scanf("%f",&b);
        printf("Enviando valores a la IP\n");
        XSuma_Set_a(&doSum,float_a_u32(a));
        XSuma_Set_b(&doSum,float_a_u32(b));
        c = u32_a_float(XSuma_Get_c_o(&doSum));
        printf("C = %f \n",c);
        XSuma_Start(&doSum);
        while(!XSuma_IsDone(&doSum));
        c = u32_a_float(XSuma_Get_c_o(&doSum));
        printf(" A + B = C \n ");
        printf(" %f + %f = %f \n ",a,b,c);
    }
    cleanup_platform();
    return 0;
}

```

Es importante realizar la conversión de float a u32 para enviar las variables a la IP esto se hace con la función “*float_a_u32*” y para obtener el resultado igualmente se tiene que realizar la conversión de u32 a float, con la función “*u32_a_float*”.

Resultados

Python

Una vez ejecutado la celda correspondiente al código principal, se ingresan los resultados, se obtendrá algo similar a la siguiente figura:

```
*****Práctica 5 ZYNQ + PYTHON *****
```

```
C = A + B
```

```
Ingrese el Valor de A17.5
```

```
Ingrese el Valor de B350.5
```

```
la suma IP tardo: 0.001129150390625 segundos
```

```
la suma es 368.0
```

```
*****Práctica 5 ZYNQ + PYTHON *****
```

```
C = A + B
```

```
Ingrese el Valor de A0.5
```

```
Ingrese el Valor de B189
```

```
la suma IP tardo: 0.0010149478912353516 segundos
```

```
la suma es 189.5
```

```
*****Práctica 5 ZYNQ + PYTHON *****
```

```
C = A + B
```

```
Ingrese el Valor de A
```

Figura 103 Resultado de la suma en Python

SDK

Una vez programado el FPGA y ejecutado el programa se obtuvo el siguiente resultado:

```

Connected to: Serial ( COM4, 115200, 0, 8 )
-----Programa para utilizar la IP - Suma
C = A + B
Donde C, A y B son datos tipo float
Ingrese valor de A :
Ingrese valor de B :
Enviando valores a la IP
C = 0.000000
A + B = C
5.500000 + 29.500000 = 35.000000
-----Programa para utilizar la IP - Suma
C = A + B
Donde C, A y B son datos tipo float
Ingrese valor de A :

```

Figura 104 Resultados de la aplicación en el puerto serial.

Práctica 6: Overlay con múltiples interrupciones.

Objetivo

- Crear una IP que realice una suma
- Crear una IP que realice una multiplicación
- Crear un diseño donde se comunique el SoC Zynq con la ip de la suma y de la multiplicación
- Realización una aplicación en Python

Desarrollo

Para esto, se genera otra IP utilizando Vitis HLS, en este caso será el de una multiplicación.

```

1 void multiplicacion (float *a, float *b, float *c)
2 {
3 #pragma HLS INTERFACE s_axilite port=return bundle=BUS_A
4     *c += *a * *b;
5 }

```

Figura 105 Código para una multiplicación en C.

Se puede retomar el mismo proyecto realizado en la sección anterior, únicamente se tiene que añadir la IP correspondiente a la multiplicación y la de Concat (este ya se encuentra en Vivado) y se realiza las conexiones resaltas en naranja:

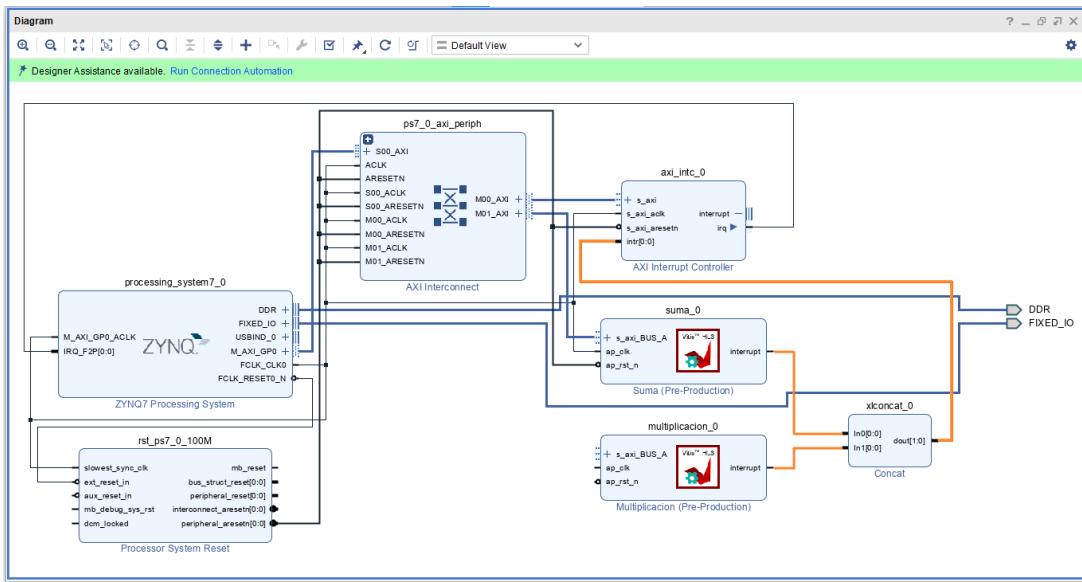


Figura 106 Conexiones necesarias para la interrupción de las dos IPs.

Después se utiliza Run Connection Automation y finalmente se obtiene el siguiente diagrama:

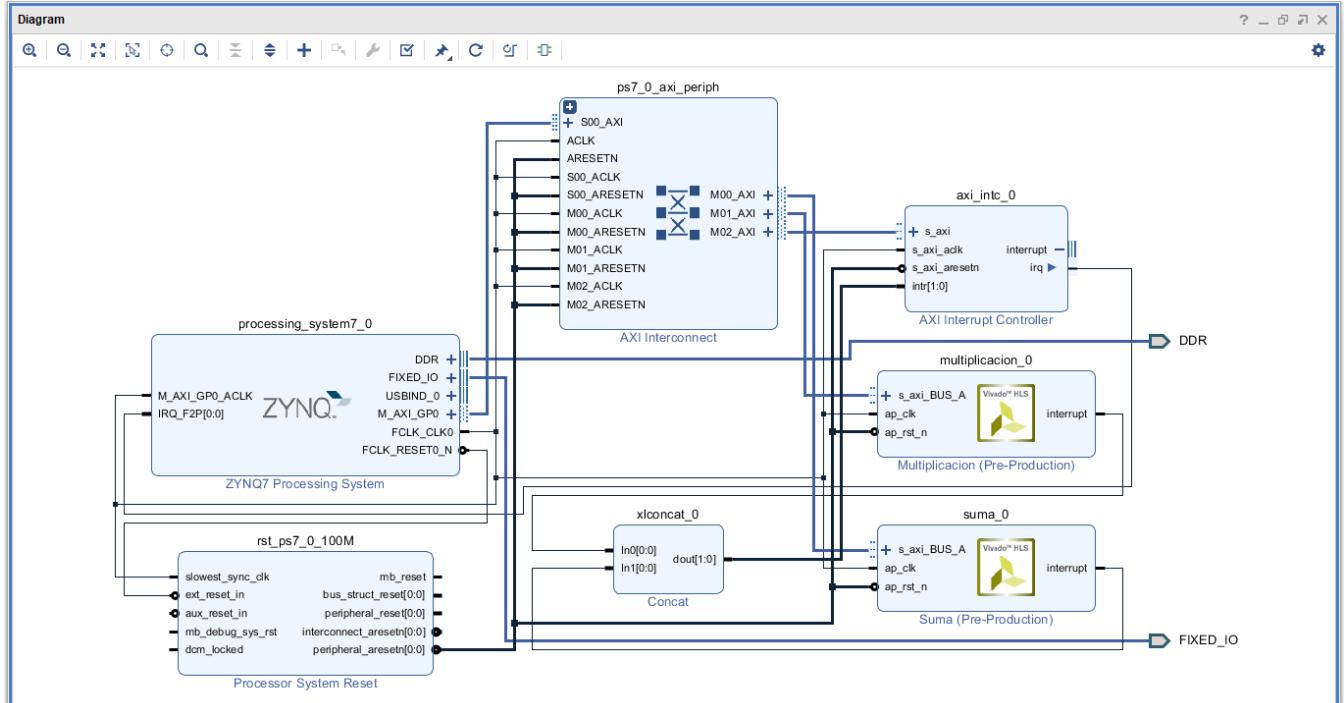


Figura 107 Diagrama completo utilizando dos IPs. generadas en Vitis HLS.

Finalmente se tiene que generar otra vez el HDL Wrapper y generar todos los archivos necesarios para programar en Python 3.

Jupyter

En esta sección se hace prácticamente lo mismo que en la práctica anterior, únicamente se añade la correspondiente consideración respecto a la multiplicación:

```

In [1]: from pynq import Overlay
        overlay = Overlay("./2IP.bit")

In [2]: overlay?

In [3]: suma = overlay.suma_0
        suma._interrupts

Out[3]: {'interrupt': {'controller': 'axi_intc_0',
                      'fullpath': 'suma_0/interrupt',
                      'index': 1}}

In [4]: multiplicacion = overlay.multiplicacion_0
        multiplicacion._interrupts

Out[4]: {'interrupt': {'controller': 'axi_intc_0',
                      'fullpath': 'multiplicacion_0/interrupt',
                      'index': 0}}

In [5]: sumIntr = suma.interrupt
        suma.register_map

Out[5]: RegisterMap {
    CTRL = Register(AP_START=0, AP_DONE=0, AP_IDLE=1, AP_READY=0, RESERVED_1=0, AUTO_RESTART=0, RESERVED_2=0),
    GIER = Register(Enable=0, RESERVED=0),
    IP_IER = Register(CHAN0_INT_EN=0, CHAN1_INT_EN=0, RESERVED=0),
    IP_ISR = Register(CHAN0_INT_ST=0, CHAN1_INT_ST=0, RESERVED=0),
    a = Register(a=0),
    b = Register(b=0),
    c_i = Register(c_i=0),
    c_o = Register(c_o=0),
    c_o_ctrl = Register(c_o_ap_vld=0, RESERVED=0)
}

In [6]: multIntr = multiplicacion.interrupt
        multiplicacion.register_map

Out[6]: RegisterMap {
    CTRL = Register(AP_START=0, AP_DONE=0, AP_IDLE=1, AP_READY=0, RESERVED_1=0, AUTO_RESTART=0, RESERVED_2=0),
    GIER = Register(Enable=0, RESERVED=0),
    IP_IER = Register(CHAN0_INT_EN=0, CHAN1_INT_EN=0, RESERVED=0),
    IP_ISR = Register(CHAN0_INT_ST=0, CHAN1_INT_ST=0, RESERVED=0),
    a = Register(a=0),
    b = Register(b=0),
    c_i = Register(c_i=0),
}

```

Figura 108 Código para exportar y verificar los atributos de las IPs.

También se crea una nueva función para la multiplicación y se utiliza `asyncio.gather` para que la suma y multiplicación se lleven a cabo de forma concurrente:

```

1. import asyncio
2. import time
3. import struct
4. import csv
5. import sys
6. from ctypes import *
7.
8. def main():
9.     menu()
10.
11.    def menu():
12.        print("*****Práctica 6 ZYNQ + PYTHON *****")

```

```

13.         print("***** Suma y Multiplicación *****")
14.         print()
15.         print("          C = A + B      \n")
16.         print("          D = A + B      \n")
17.         valorA = float(input(" Ingrese el Valor de A"))
18.         valorB = float(input(" Ingrese el Valor de B"))
19.         A=float_to_hex(valorA)
20.         B=float_to_hex(valorB)
21.         loop = asyncio.get_event_loop()
22.         todo = asyncio.gather(sumaconIP(A,B), multconIP(A,B))
23.         loop.run_until_complete(asyncio.gather(todo))
24.         loop.close()
25.
26.     def hex_to_float(s):
27.         i = int(s, 16)
28.         cp = pointer(c_int(i))
29.         fp = cast(cp, POINTER(c_float))
30.         return fp.contents.value
31.
32.     def float_to_hex(value):
33.         return struct.pack('f', value)
34.
35.     async def sumaconIP(A,B):
36.         inicio=time.time()
37.         suma.write(0x004,1)
38.         suma.write(0x008,1)
39.         suma.write(0x010,A)
40.         suma.write(0x018,B)
41.         suma.write(0x000,1)
42.         await sumIntr.wait()
43.         final=time.time()
44.         a=suma.read(0x028)
45.         c = hex_to_float(hex(a))
46.         total=final-inicio
47.         print(f"la suma IP tardó: {total} segundos")
48.         print(f"la suma es {c}")
49.         suma.write(0x000,128)
50.
51.     async def multconIP(A,B):
52.         inicio=time.time()
53.         multiplicacion.write(0x004,1)
54.         multiplicacion.write(0x008,1)
55.         multiplicacion.write(0x010,A)
56.         multiplicacion.write(0x018,B)
57.         multiplicacion.write(0x000,1)
58.         await multIntr.wait()
59.         final=time.time()
60.         a=multiplicacion.read(0x028)
61.         d = hex_to_float(hex(a))

```

```

62.         totals=finals-inicios
63.         print(f"la multiplicación tardo: {totals} segundos")
64.         print(f"D = {d}")
65.         suma.write(0x000,128)
66.
67.     while 1:
68.         main()

```

Resultados

Al ejecutar el código se obtengo el siguiente resultado:

```

*****Práctica 6 ZYNQ + PYTHON ****
*****      Suma y Multiplicación      *****

        C = A + B
        D = A + B

Ingrese el Valor de A7.5
Ingrese el Valor de B6.5
la multiplicación tardo: 0.0010139942169189453 segundos
D = 48.75
la suma IP tardo: 0.003908872604370117 segundos
la suma es 14.0
*****Práctica 6 ZYNQ + PYTHON ****
*****      Suma y Multiplicación      *****

        C = A + B
        D = A + B

Ingrese el Valor de A 

```

Figura 109 Resultado del código en Python

Práctica 7: Suma de Matrices

Objetivo

- Crear una IP que realice una suma de 2 dos matrices de 3x3
- Crear un diseño donde se comunique el SoC Zynq con la ip de la suma de matrices
- Realizar una aplicación en Python

Desarrollo

Para realizar esta práctica se seguirá el siguiente proceso:



Vitis HLS

Para la IP se propone utilizar la interfaz BRAM, el cual implementa argumentos de matriz como una interfaz RAM estándar, se optó por utilizar BRAM por cada matriz. El código utilizado es el siguiente:

```

#include "mult.h"

void SumaMat(const dtype2 A[DIM], const dtype2 B[DIM], dtype2 C[DIM])
{
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS

L1:for (int i = 0; i < 9; ++i)
{
    C[i] = A[i] + B[i];
}

}

```

Se utilizaron las siguientes directivas:

```

▼ [●] SumaMat
  # HLS INTERFACE s_axilite port=return bundle=CRTL_BUS
  ● A
  % HLS INTERFACE bram port=A
  ● B
  % HLS INTERFACE bram port=B
  ● C
  % HLS INTERFACE bram port=C
  ▼ [XY] L1
    % HLS PIPELINE

```

Figura 110 Directivas utilizadas en la suma de Matrices.

Posteriormente se realizó la síntesis y se exporto la IP.

Vivado

Una vez creado el proyecto, se tiene que agregar la IP generada en la sección anterior. En el diseño se usarán las siguientes IP's:

- Procesador Zynq
- IP generada en la sección anterior.
- Block Memory Generator (Una por cada matriz)
- AXI BRAM Controller (Una por cada matriz)

La IP AXI BRAM Controller se utiliza para el SoC Zynq se pueda comunicar con el bloque de la memoria, debe estar configurado teniendo una sola interfaz:

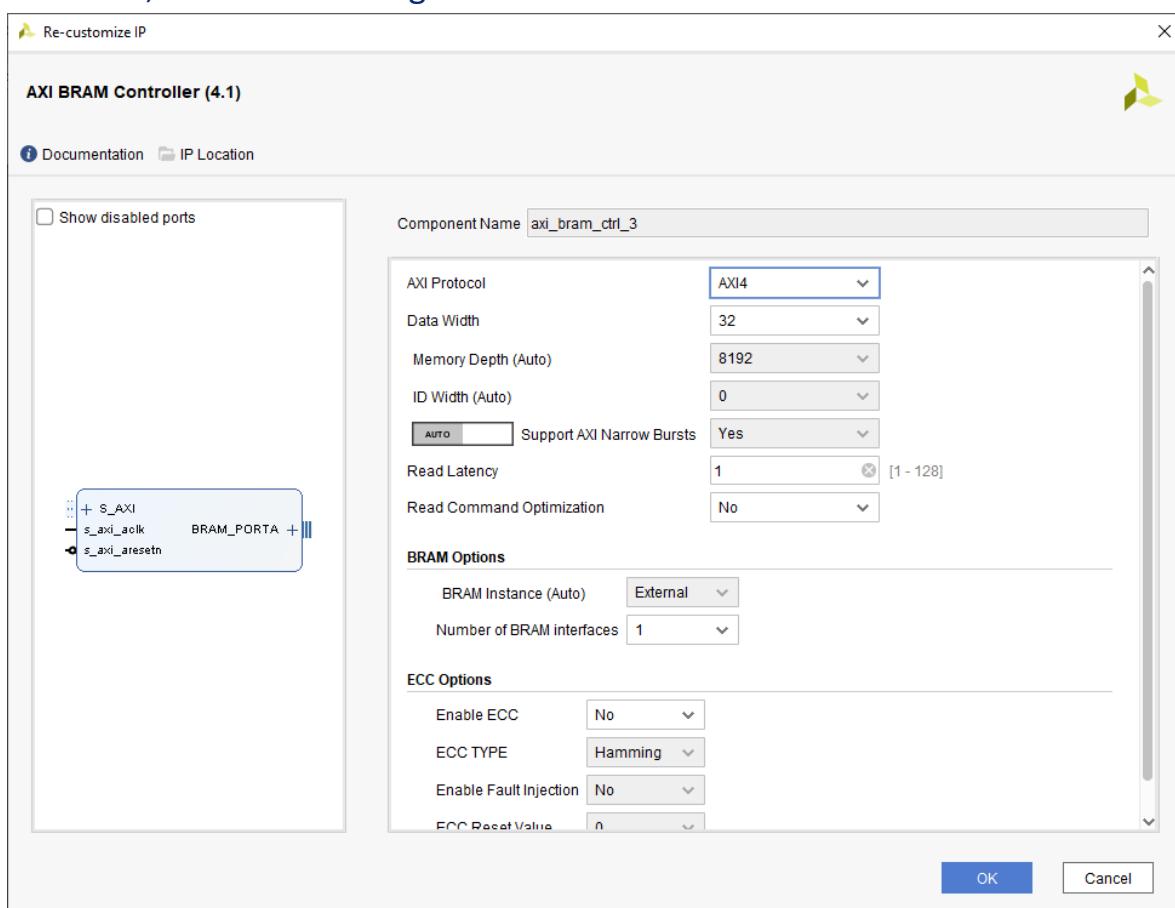


Figura 111 AXI BRAM CONTROLLER (Una interfaz).

Y para la IP Block Memory Generator, el tipo de memoria debe ser “True Dual Port RAM”:

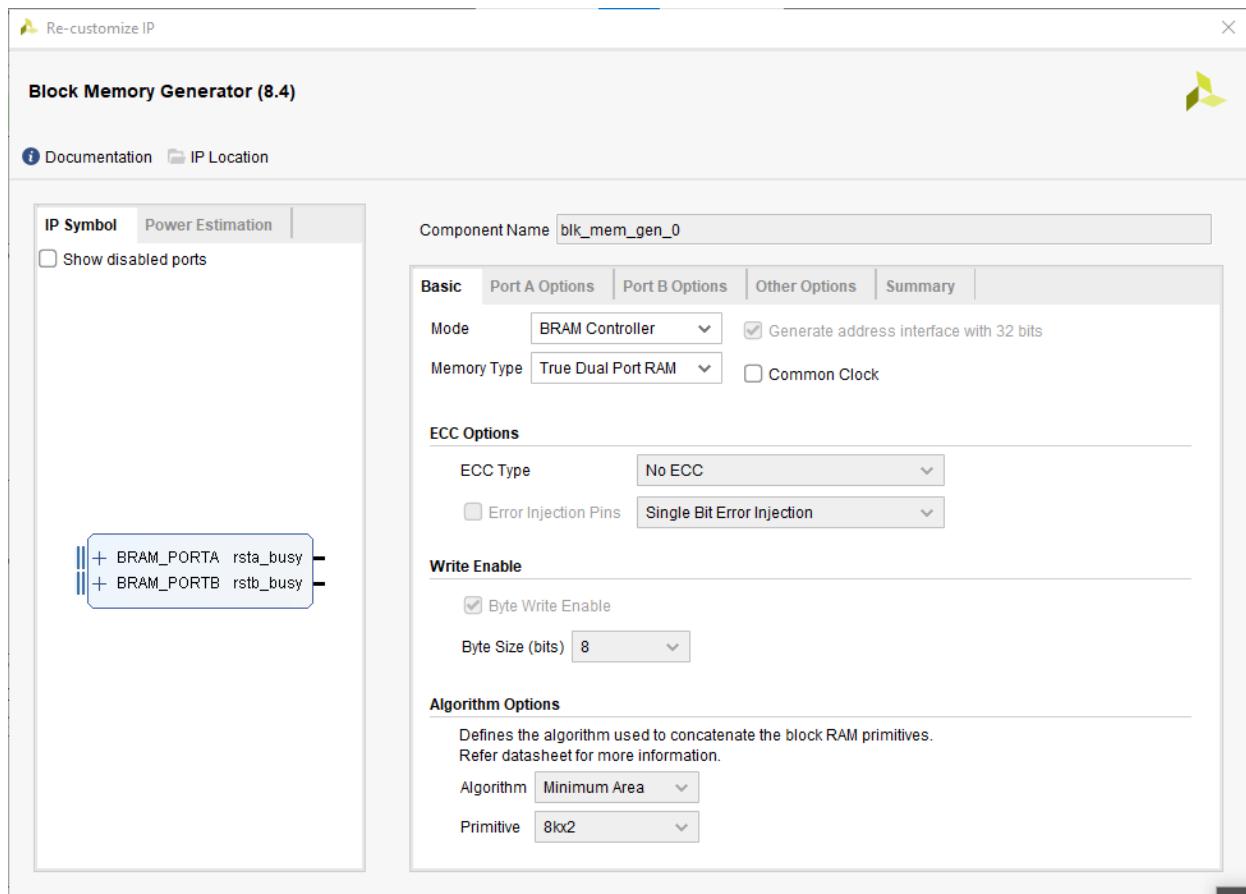


Figura 112 Configuración para la IP Block Memory Generator.

El diagrama una vez realizado las conexiones debe lucir de la siguiente manera:

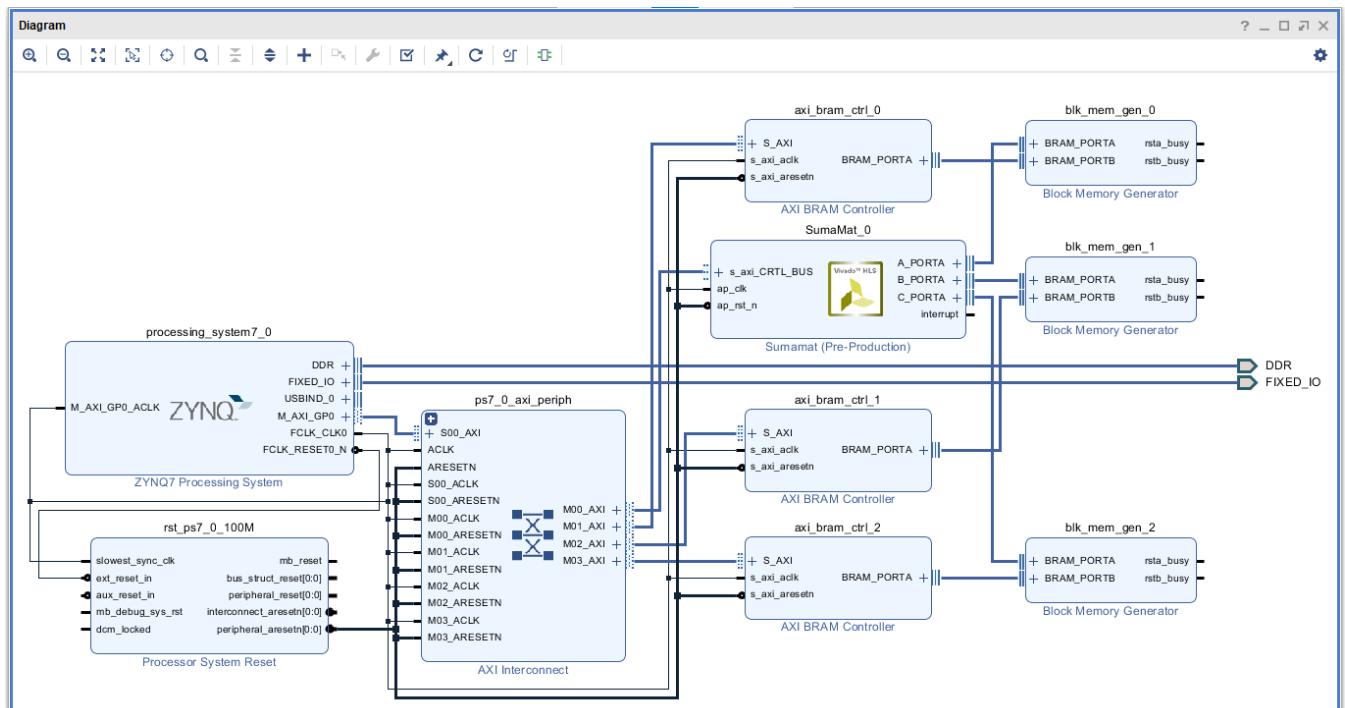


Figura 113 Diagrama utilizado para la suma de matrices.

También es importante revisar las direcciones asignadas a cada bloque de memoria, ya posteriormente se utilizará al momento de realizar la aplicación.

Address Editor						
Cell	Slave Interface	Base N...	Offset Address	Range	High Address	
processing_system7_0						
Data (32 address bits : 0x40000000 [1G])						
SumaMat_0	s_axi_CRTL_B...	Reg	0x43C0_0000	64K	0x43C0_FFFF	
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF	
axi_bram_ctrl_1	S_AXI	Mem0	0x4200_0000	8K	0x4200_1FFF	
axi_bram_ctrl_2	S_AXI	Mem0	0x4400_0000	8K	0x4400_1FFF	

Figura 114 Direcciones de los bloques de memoria.

Posteriormente se crea el HDL Wrapper y se genera el bitstream.

Jupyter

Se utiliza la clase MMIO, la cual permite que Python tenga acceso a la memoria del sistema, en particular con los registros y espacios de direcciones de los periféricos. Por lo tanto, se declaran las direcciones para cada matriz de acuerdo con el Address Editor de Vivado.

```
from pynq import MMIO
base_addressA = 0x40000000
base_addressB = 0x42000000
base_addressC = 0x44000000
mem_size = 4096
mmioA = MMIO(base_addressA, mem_size)
mmioB = MMIO(base_addressB, mem_size)
mmioC = MMIO(base_addressC, mem_size)
```

Lo siguientes es crear una clase Overlay, descargarla a la tarjeta y verificar si está cargada:

```
from pynq import Overlay
overlay = Overlay("design_1.bit")
overlay.download()
overlay.is_loaded()
```

Finalmente se crea lo referente a la aplicación, al igual que en la practica 6, se utiliza un menú y las funciones para convertir de hexadecimal a float y viceversa, el código propuesto es el siguiente:

```
from ctypes import *
import struct
import ipywidgets as widgets

received_dataC = [0]*9
```

```

C_FLOAT = [0]*9
received_dataA = [0]*9
A_FLOAT = [0]*9
received_dataB = [0]*9
B_FLOAT = [0]*9

def main():
    menu()

def menu():
    address_offset = 0
    address_offset2 = 0
    print("*****Práctica 7 ZYNQ + PYTHON *****")
    print("*****      Suma de Matrices      *****")
    print()
    print("          C = A + B          \n")
    print("Matriz A \n")
    for i in range(3):
        for j in range(3):
            elemento = float(input())
            aa = float_to_hex(elemento)
            mmioA.write(address_offset,aa)
            address_offset = address_offset + 4
    print("Matriz B \n")
    for i in range(3):
        for j in range(3):
            elemento = float(input())
            bb = float_to_hex(elemento)
            mmioB.write(address_offset2,bb)
            address_offset2 = address_offset2 + 4
    suma.write(0x00, 4)
    suma.write(0x00, 1)
    for i in range(0,9):
        received_dataA[i] = mmioA.read(i*4)
        A_FLOAT[i] = hex_to_float(hex(received_dataA[i]))
        received_dataB[i] = mmioB.read(i*4)
        B_FLOAT[i] = hex_to_float(hex(received_dataB[i]))
        received_dataC[i] = mmioC.read(i*4)
        C_FLOAT[i] = hex_to_float(hex(received_dataC[i]))
    print(" Matriz A: \n")
    imprimirMAtriz(A_FLOAT)
    print(" Matriz B: \n")
    imprimirMAtriz(B_FLOAT)
    print(" Matriz C: \n")
    imprimirMAtriz(C_FLOAT)

def hex_to_float(s):
    i = int(s, 16)
    cp = pointer(c_int(i))
    fp = cast(cp, POINTER(c_float))
    return fp.contents.value

def float_to_hex(value):
    return struct.pack('f', value)

def imprimirMAtriz(matriz):
    print(f"{matriz[0]} {matriz[1]} {matriz [2]}")

```

```
print(f'{matriz[3]} {matriz[4]} {matriz[5]}')
print(f'{matriz[6]} {matriz[7]} {matriz[8]}')

while 1:
    main()
```

Resultado

Al ejecutar la aplicación, se obtuvo el siguiente resultado:

Figura 115 Resultado de la suma de matrices.

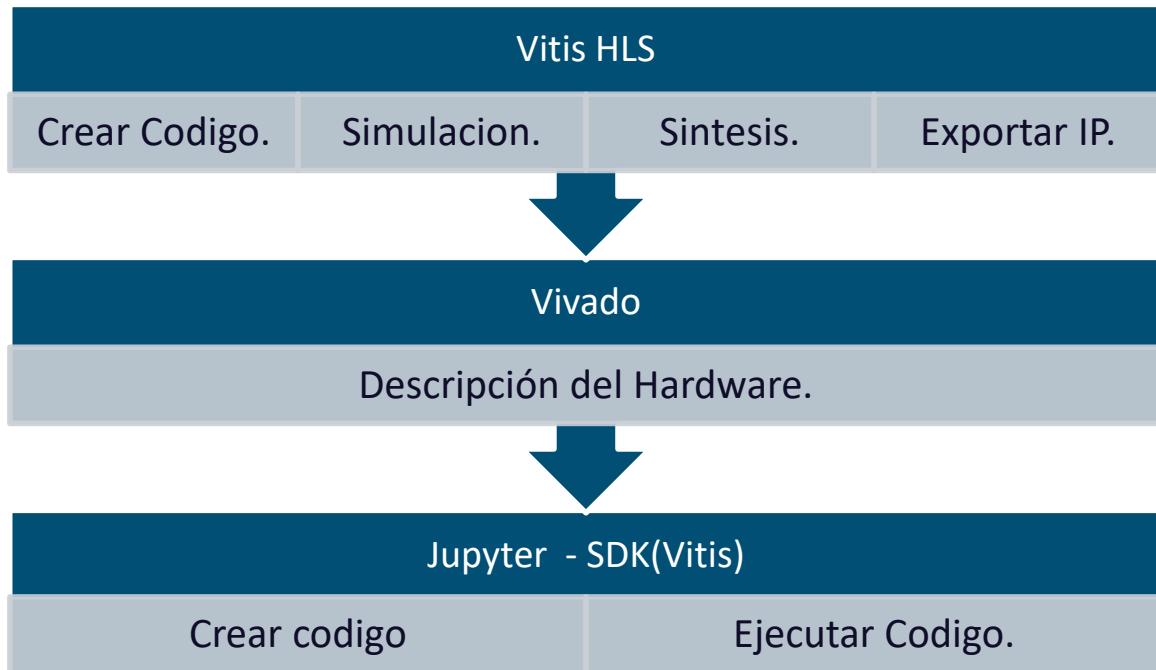
Práctica 8: Multiplicación de Matrices

Objetivo

- Crear una IP que realice una multiplicación de matrices
- Crear un diseño donde se comunique el SoC Zynq con la ip de la multiplicación de matrices
- Realizar una aplicación en SDK/Vitis

Desarrollo

Para el desarrollo de esta práctica se realizará el siguiente proceso:



Vitis HLS

Para generar la IP, se propone el siguiente código:

```

#include "multiplicacion.h"

void matrizMult(const dtype2 A[DIM][DIM], const dtype2
B[DIM][DIM], dtype2 C[DIM][DIM])
{
#pragma HLS INTERFACE s_axilite port=return bundle=CRTL_BUS

dtype2 sum;

// matrix multiplication of a A*B matrix
L1:for (int ia = 0; ia < DIM; ++ia)
{
L2:for (int ib = 0; ib < DIM; ++ib)
{
sum = 0;
L3:for (int id = 0; id < DIM; ++id)
{
sum += A[ia][id] * B[id][ib];
}
C[ia][ib] = sum;
}
}
}

}

```

Y se utilizaron las siguientes directivas:

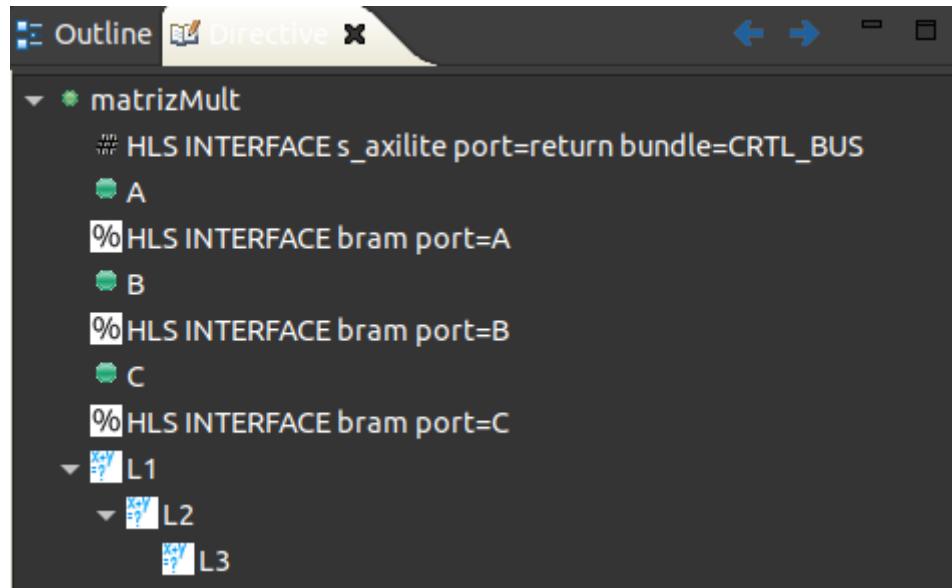


Figura 116 Directivas para la multiplicación de matrices.

La interfaz BRAM implementa argumentos de matriz como una interfaz RAM estándar.

Para el TestBench se propone el siguiente código:

```

#include <iostream>
#include "multiplicacion.h"
using namespace std;
int main ()
{
    dtype2 num=1;
    dtype2 a[DIM] [DIM];
    dtype2 b[DIM] [DIM];
    dtype2 c[DIM] [DIM];
    for (int i = 0;i<DIM;++i){
        for (int j = 0;j<DIM;++j) {
            a[i][j]=num+0.015;
            num=num+1;
            if (i==j)
            {
                b[i][j]=1;
            }else{
                b[i][j]=0;
            }
        }
    }
    //imprimir matriz
    cout <<"Matriz A"<< endl;
    for (int i=0;i<DIM;++i){
        for (int j=0;j<DIM;++j) {
            cout << a[i][j] <<" ";
        }
        cout << endl;
    }
    cout <<"Matriz B"<< endl;
    for (int i=0;i<DIM;++i){
        for (int j=0;j<DIM;++j) {
            cout << b[i][j] <<" ";
        }
        cout << endl;
    }
    matrizMult(a,b,c);
    cout <<"Matriz c"<< endl;
    for (int i=0;i<DIM;++i){
        for (int j=0;j<DIM;++j) {
            cout << c[i][j] <<" ";
        }
        cout << endl;
    }
    return 0;
}

```

Una vez ejecutado la simulación se obtiene el siguiente resultado:

```
1 INFO: [SIM 2] ***** CSIM start *****
2 INFO: [SIM 4] CSIM will launch GCC as the compiler.
3   Compiling ../../../../testbench/prueba.cpp in debug mode
4   Compiling ../../../../multiplicacion.cpp in debug mode
5   Generating csim.exe
6 Matriz A
7 1.015 2.015 3.015
8 4.015 5.015 6.015
9 7.015 8.015 9.015
10 Matriz B
11 1 0 0
12 0 1 0
13 0 0 1
14 Matriz C
15 1.015 2.015 3.015
16 4.015 5.015 6.015
17 7.015 8.015 9.015
18 INFO: [SIM 1] CSim done with 0 errors.
19 INFO: [SIM 3] ***** CSIM finish *****
```

Figura 117 Resultado de la simulación (Multiplicación de matrices).

Finalmente se realiza la síntesis de la solución y se exporta la IP.

Vivado

Para el diseño se agregan las siguientes IP's:

- Zynq
 - Block Memory Generator: Es la memoria donde se almacenarán las matrices.
 - AXI BRAM Controller: Permite al procesador Zynq acceder a las memorias, tanto para leerlas y escribir en ellas.

El diagrama final es el siguiente:

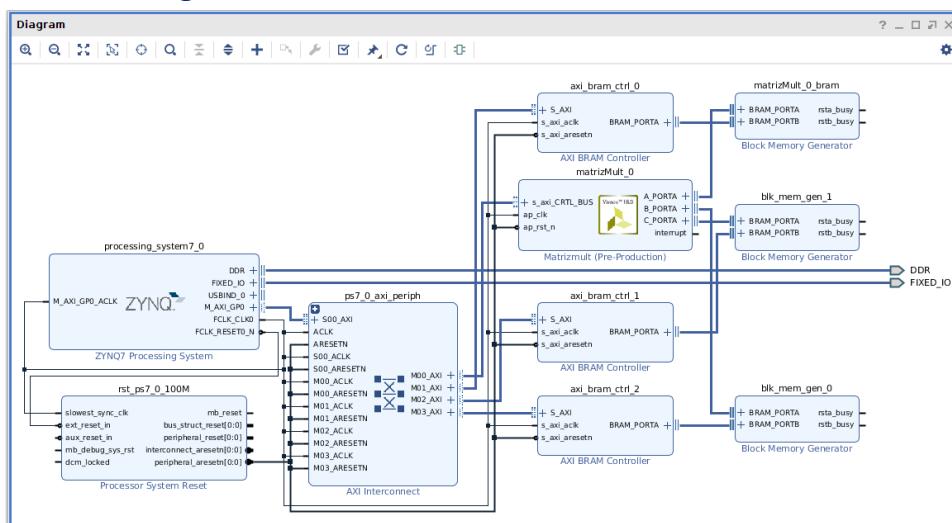


Figura 118 Diagrama para la multiplicación de matrices.

SDK o Vitis

El código en C, prácticamente sigue la misma estructura que en Python, se le asigna un apuntador a la dirección de cada memoria, se tiene una función para iniciar la IP. La diferencia más importante es la facilidad de almacenar y leer los datos, ya que no se requiere realizar ninguna conversión. El código propuesto es el siguiente:

```
#include <stdio.h>
#include "platform.h"
#include <xmatrixmult.h>
#include <math.h>

#include "xil_printf.h"
```

```

float *A = (float *)0x40000000;
float *B = (float *)0x42000000;
float *C = (float *)0x44000000;

XMatrizmult hacerMultiplicacion;
XMatrizmult_Config *hacerMultiplicacion_cfg;

void iniciar_matrizMult(){
int status = 0;
hacerMultiplicacion_cfg = XMatrizmult_LookupConfig(XPAR_MATRIZMULT_0_DEVICE_ID);
if (hacerMultiplicacion_cfg){
status = XMatrizmult_CfgInitialize(&hacerMultiplicacion,hacerMultiplicacion_cfg);
if (status != XST_SUCCESS)
{
printf("Ha ocurrido un error al iniciar la IP(multiplicacion de matrices)");
}
}
}

int main()
{
    init_platform();
    iniciar_matrizMult();
    while(1){
printf("-----Practica 7-----\n");
printf("-----Multiplicacion de Matrices---\n");
printf("          C=AB\n");
printf(" Matriz A \n");
for (int i = 0;i<9;++i){
    printf(" A %d : \n",i+1);
    scanf("%f",&A[i]);

}
printf(" Matriz B \n");
for (int i = 0;i<9;++i){
    printf(" B %d : \n",i+1);
    scanf("%f",&B[i]);
}

}

//UTILIZANDO IP
XMatrizmult_Start(&hacerMultiplicacion);
while(!XMatrizmult_IsDone(&hacerMultiplicacion));

printf("MAtriz A: \n");
printf("%f %f %f \n",A[0],A[1],A[2]);
printf("%f %f %f \n",A[3],A[4],A[5]);
printf("%f %f %f \n",A[6],A[7],A[8]);
printf("MAtriz B: \n");
printf("%f %f %f \n",B[0],B[1],B[2]);
printf("%f %f %f \n",B[3],B[4],B[5]);
printf("%f %f %f \n",B[6],B[7],B[8]);
printf("MAtriz C (resultado): \n");
printf("%f %f %f \n",C[0],C[1],C[2]);
printf("%f %f %f \n",C[3],C[4],C[5]);

```

```

    printf("%f %f %f \n",C[6],C[7],C[8]);
}
cleanup_platform();
return 0;
}

```

Resultados

Una vez programado el FPGA y ejecutado la aplicación, se obtuvo el siguiente funcionamiento en la terminal serial:

```

-----Practica 7-----
-----Multiplicacion de Matrices-----
C=AB
Matriz A
A 1:
A 2:
A 3:
A 4:
A 5:
A 6:
A 7:|
A 8:
A 9:
Matriz B
B 1:
B 2:
B 3:
B 4:
B 5:
B 6:
B 7:
B 8:
B 9:
MAtriz A:
1.000000 2.000000 2.500000
6.500000 3.000000 4.000000
5.000000 6.000000 7.000000
MAtriz B:
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
Matriz C (resultado):
26.500000 32.000000 37.500000
46.500000 60.000000 73.500000
78.000000 96.000000 114.000000

```

Figura 119 Resultado de la aplicación en el puerto serial

Practica 9: Zynq con AXI TIMER

Objetivo

- Crear un diseño donde se utilice la IP AXI TIMER
- Crear una aplicación en Vitis/SDK
- Crear una aplicación en Python

Desarrollo

Vivado

Para realizar el diseño se utilizará de base el realizado en la práctica 8, sin embargo, esto es opcional, se añade la IP y se configura de la siguiente manera:

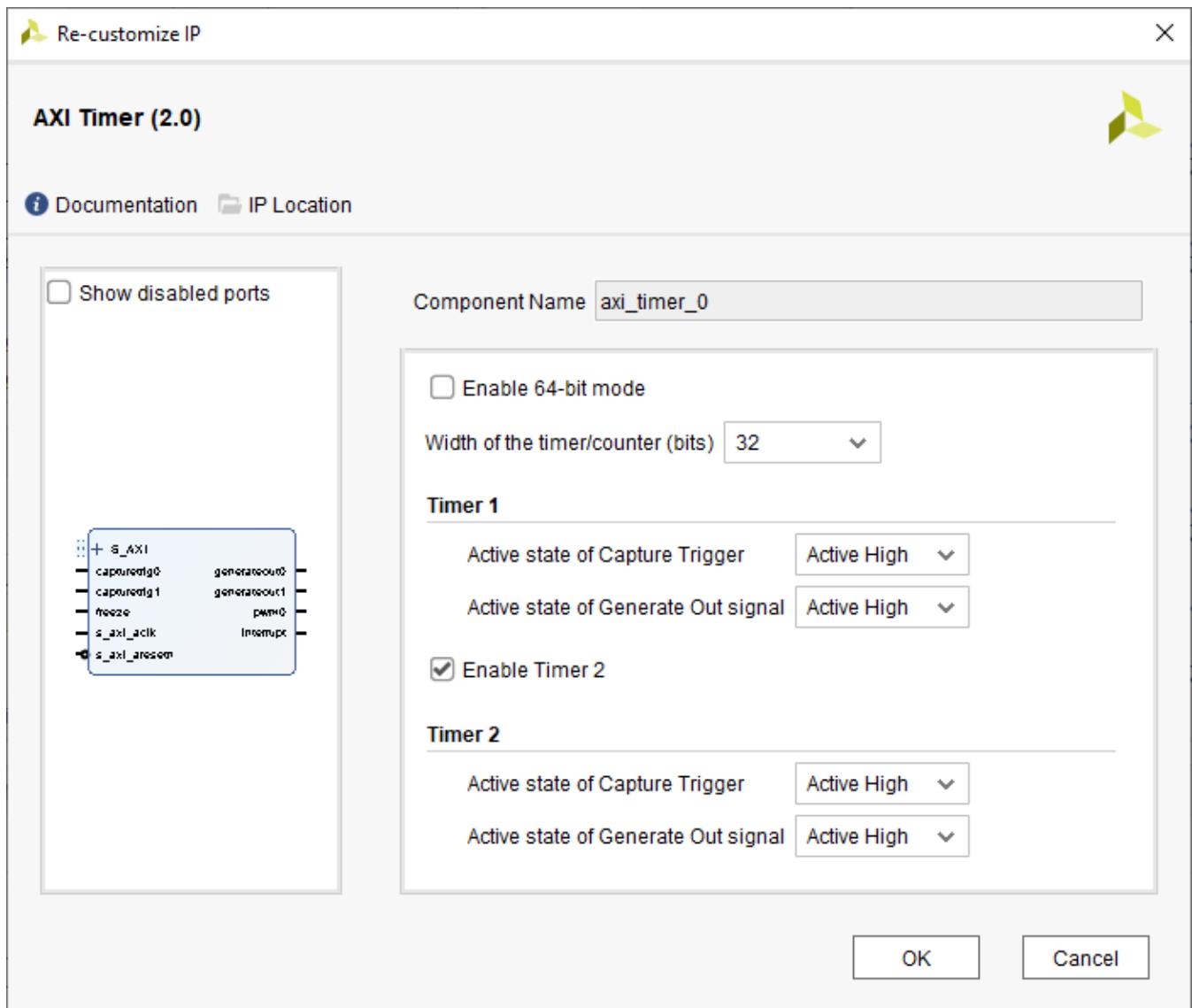


Figura 120 Configuración del AXI TIMER.

Es importante recalcar que se utilizaran 32 bits porque posteriormente se hará referencia a ello. Una vez hecho esto, se utiliza “Run Connection Automation” y el diagrama final es el siguiente:

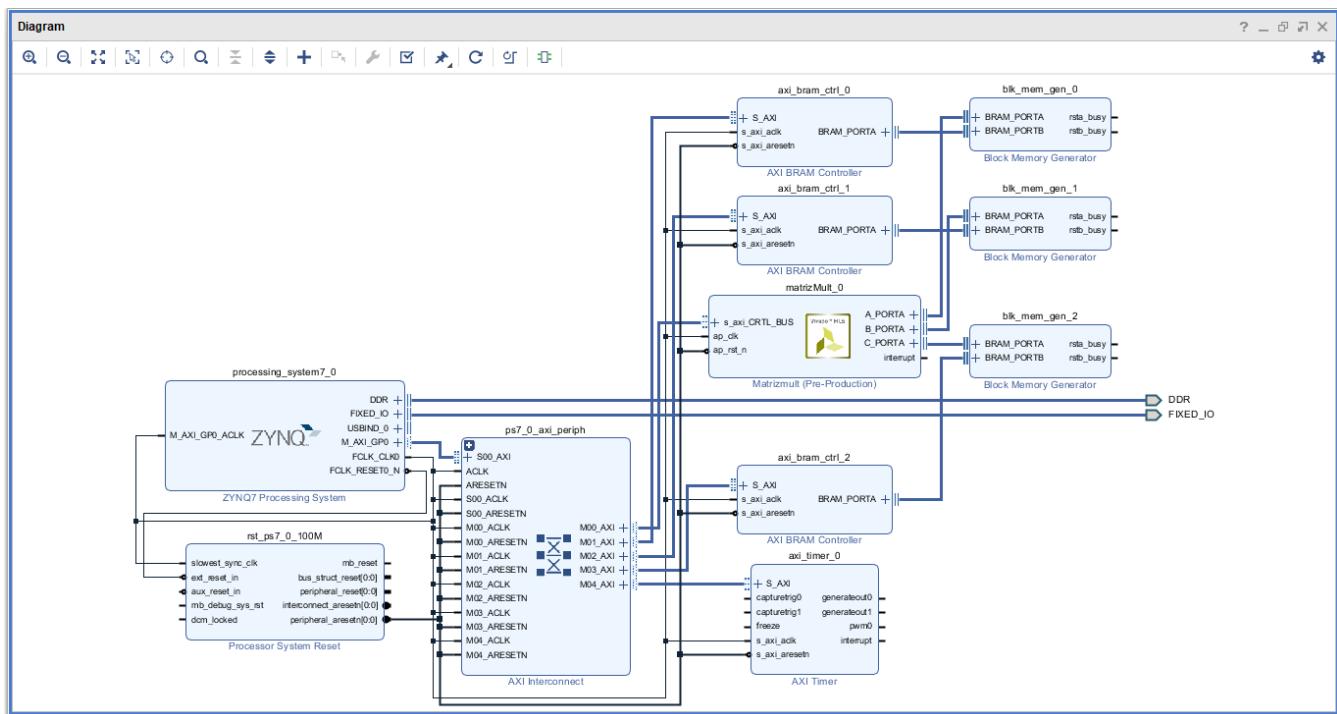


Figura 121 Diagrama para utilizar el AXI TIMER.

Vitis/SDK

La aplicación se realiza en C++ sin embargo es de mucha utilidad crear una aplicación en C para poder copiar algunos archivos. En la subcarpeta “src” del proyecto en C++, se agrega un nuevo archivo header:

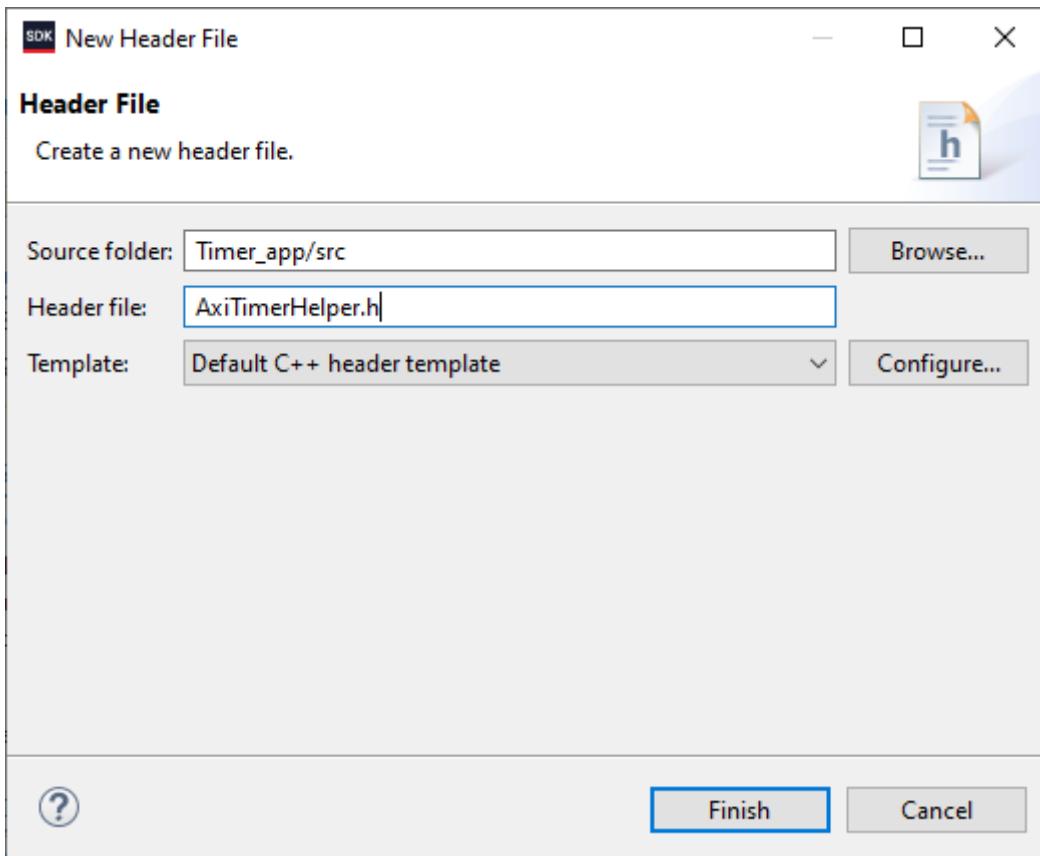


Figura 122 Archivo Header para la aplicación.

El archivo de cabecera se modifica de la siguiente manera:

```
/*
 * AxiTimerHelper.h
 *
 * Created on: 15/04/2021
 * Author: Claudio
 */
```

```
#ifndef SRC_AXITIMERHELPER_H_
#define SRC_AXITIMERHELPER_H_

#include "xil_types.h"
#include "xtmrctr.h"
#include "xparameters.h"

class AxiTimerHelper{
public:
    AxiTimerHelper();
    virtual ~AxiTimerHelper();
    unsigned int getElapsedTicks();
    double getElapsedTimerInSeconds();
    unsigned int startTimer();
    unsigned int stopTimer();
    double getClockPeriod();
    double getTimerClockFreq();
private:
    XTmrCtr m_AxiTimer;
    unsigned int m_tickCounter1;
    unsigned int m_tickCounter2;
    double m_clockPeriorSeconds;
    double m_timerClockFreq;
};
```

```
#endif /* SRC_AXITIMERHELPER_H_ */
```

También se añade un nuevo archivo fuente y se copia el siguiente código:

```
#include "AxiTimerHelper.h"

AxiTimerHelper::AxiTimerHelper() {
    // TODO Auto-generated constructor stub
    //INICIANDO EL HARDWARE
    XTmrCtr_Initialize(&m_AxiTimer, XPAR_TMRCTR_0_DEVICE_ID);

    //Obtener el periodo del reloj en segundos
    m_timerClockFreq = (double) XPAR_AXI_TIMER_0_CLOCK_FREQ_HZ;
    m_clockPeriorSeconds = (double) 1/m_timerClockFreq;
}

AxiTimerHelper::~AxiTimerHelper() {
    // TODO Auto-generated destructor stub
}

unsigned int AxiTimerHelper::getElapsedTicks() {
    return m_tickCounter2 - m_tickCounter1;
}

unsigned int AxiTimerHelper::startTimer() {
    //tiempo de inicio del timer 0
```

```

XTmrCtr_Reset(&m_AxiTimer, 0);
m_tickCounter1 = XTmrCtr_GetValue(&m_AxiTimer, 0);
XTmrCtr_Start(&m_AxiTimer, 0);
return m_tickCounter1;
}

unsigned int AxiTimerHelper::stopTimer() {
    XTmrCtr_Stop(&m_AxiTimer, 0);
    m_tickCounter2 = XTmrCtr_GetValue(&m_AxiTimer, 0);
    return m_tickCounter2 - m_tickCounter1;
}

double AxiTimerHelper::getElapsedTimerInSeconds() {
    double elapsedTimeInSeconds = (double)(m_tickCounter2 -
m_tickCounter1) * m_clockPeriodSeconds;
    return elapsedTimeInSeconds;
}

double AxiTimerHelper::getClockPeriod() {
    return m_clockPeriodSeconds;
}

double AxiTimerHelper::getTimerClockFreq() {
    return m_timerClockFreq;
}

```

Finalmente se copian los siguientes archivos desde el proyecto en C:

- platform_config.h
- platform.h
- platform.c

Para el archivo “platform.c” es necesario cambiar la extensión a “.cc”. Finalmente, el archivo principal es el siguiente:

```

#include <stdio.h>
#include "platform.h"
#include <xmatrixmult.h>
#include "xil_printf.h"
#include "AxiTimerHelper.h"
#include "sleep.h"

#include "xil_printf.h"

float *A = (float *)0x40000000;
float *B = (float *)0x42000000;
float *C = (float *)0x44000000;

XMatrixmult hacerMultiplicacion;
XMatrixmult_Config *hacerMultiplicacion_cfg;

void iniciar_matrixMult() {
int status = 0;

```

```

hacerMultiplicacion_cfg =
XMatrixmult_LookupConfig(XPAR_MATRIXMULT_0_DEVICE_ID);
if (hacerMultiplicacion_cfg) {
status =
XMatrixmult_CfgInitialize(&hacerMultiplicacion,hacerMultiplicacion_cfg);
if (status != XST_SUCCESS)
{
printf("Ha ocurrido un error al iniciar la IP(multiplicacion de matrices)");
}
}
}

int main()
{
    init_platform();
    iniciar_matrizMult();
    AxiTimerHelper myTimer;
    while(1){
printf("-----Practica 7-----\n");
printf("-----Multiplicacion de Matrices---\n");
printf("          C=AB\n");
printf(" Matriz A \n");
for (int i = 0;i<9;++i){
    printf(" A %d : \n",i+1);
    scanf("%f",&A[i]);
}

printf(" Matriz B \n");
for (int i = 0;i<9;++i){
    printf(" B %d : \n",i+1);
    scanf("%f",&B[i]);
}

}

//UTILIZANDO IP
myTimer.startTimer();
XMatrixmult_Start(&hacerMultiplicacion);
while(!XMatrixmult_IsDone (&hacerMultiplicacion));
myTimer.stopTimer();
printf("Realizado en %f segundos\n",myTimer.getElapsedTimerInSeconds ());
printf("MAtriz A: \n");
printf("%f %f %f \n",A[0],A[1],A[2]);
printf("%f %f %f \n",A[3],A[4],A[5]);
printf("%f %f %f \n",A[6],A[7],A[8]);
printf("MAtriz B: \n");
printf("%f %f %f \n",B[0],B[1],B[2]);
printf("%f %f %f \n",B[3],B[4],B[5]);
printf("%f %f %f \n",B[6],B[7],B[8]);
printf("Matriz C (resultado): \n");
printf("%f %f %f \n",C[0],C[1],C[2]);
printf("%f %f %f \n",C[3],C[4],C[5]);
printf("%f %f %f \n",C[6],C[7],C[8]);
}
cleanup_platform();

```

```

    return 0;
}

```

Las funciones importantes son las siguientes:

- myTimer.startTimer();
- myTimer.stopTimer()
- myTimer.getElapsedTimerInSeconds()

Python

Para Python, prácticamente es lo mismo que en la práctica 8, solo se modificó el código principal para añadir las funciones respecto al AXI TIMER:

```

from ctypes import *
import struct
import time

AXI_CLOCK_PERIOD = 1/100000000;
MAX_COUNT = 4294967295;
received_dataC = [0]*9
C_FLOAT = [0]*9
received_dataA = [0]*9
A_FLOAT = [0]*9
received_dataB = [0]*9
B_FLOAT = [0]*9

def main():
    menu()

def menu():
    address_offset = 0
    address_offset2 = 0
    print("*****Práctica 9 ZYNQ + PYTHON *****")
    print("*****      Suma de Matrices      *****")
    print()
    print("          C = AB          \n")
    print("Matriz A \n")
    for i in range(3):
        for j in range(3):
            elemento = float(input())
            aa = float_to_hex(elemento)
            mmioA.write(address_offset,aa)
            address_offset = address_offset + 4
    print("Matriz B \n")
    for i in range(3):
        for j in range(3):
            elemento = float(input())
            bb = float_to_hex(elemento)
            mmioB.write(address_offset2,bb)
            address_offset2 = address_offset2 + 4
    multiplicacion.write(0x00, 4)
    IniciarTimer()
    multiplicacion.write(0x00, 1)
    esperarResultado()
    TLRx=DetenerTimer()
    TIMING_INTERVAL = (MAX_COUNT - TLRx + 2) * AXI_CLOCK_PERIOD;

```

```

print("el tiempo fue ", TIMING_INTERVAL)
for i in range(0,9):
    received_dataA[i] = mmioA.read(i*4)
    A_FLOAT[i] = hex_to_float(hex(received_dataA[i]))
    received_dataB[i] = mmioB.read(i*4)
    B_FLOAT[i] = hex_to_float(hex(received_dataB[i]))
    received_dataC[i] = mmioC.read(i*4)
    C_FLOAT[i] = hex_to_float(hex(received_dataC[i]))
print(" Matriz A: \n")
imprimirMAtriz(A_FLOAT)
print(" Matriz B: \n")
imprimirMAtriz(B_FLOAT)
print(" Matriz C: \n")
imprimirMAtriz(C_FLOAT)

def hex_to_float(s):
    i = int(s, 16)
    cp = pointer(c_int(i))
    fp = cast(cp, POINTER(c_float))
    return fp.contents.value

def float_to_hex(value):
    return struct.pack('f', value)

def imprimirMAtriz(matriz):
    print(f"{matriz[0]} {matriz[1]} {matriz [2]}")
    print(f"{matriz[3]} {matriz[4]} {matriz [5]}")
    print(f"{matriz[6]} {matriz[7]} {matriz [8]}")

def esperarResultado():
    while True:
        estado = multiplicacion.read(0x00)
        if (estado == 2 or estado == 4 or estado == 6):
            break

def IniciarTimer():
    timer.register_map.TLR0 = 4294967295
    timer.register_map.TCSR0.LOAD0 = 1
    timer.register_map.TCSR0.LOAD0 = 0
    timer.register_map.TCSR0.T0INT = 0
    timer.register_map.TCSR0.MDT0 = 0
    timer.register_map.TCSR0.UDT0 = 1
    timer.register_map.TCSR0.ENT0 = 1

def DetenerTimer():
    timer.register_map.TCSR0.ENT0 = 0
    timer.register_map.TCSR1.ENT1 = 0
    a = timer.read(0x08)
    return a

while 1:
    main()

```

El AXI TIMER básicamente es un contador, en este caso se configuro para que sea un contador ascendente por lo que, de acuerdo con la documentación de la IP, se tiene que realizar la siguiente operación:

When the counter is set to count up,

```
TIMING_INTERVAL = (MAX_COUNT - TLRx + 2) * AXI_CLOCK_PERIOD
```

where MAX_COUNT is the maximum count value of the counter, such as 0xFFFFFFFF for a 32-bit counter.

Para más información es recomendable revisar la documentación.

Resultados

Python

```
*****Práctica 9 ZYNQ + PYTHON ****
***** Suma de Matrices *****
```

```
C = AB
```

```
Matriz A
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

```
Matriz B
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9
```

```
el tiempo fue 0.00029912
```

```
Matriz A:
```

```
1.0 2.0 3.0  
4.0 5.0 6.0  
7.0 8.0 9.0
```

```
Matriz B:
```

```
1.0 2.0 3.0  
4.0 5.0 6.0  
7.0 8.0 9.0
```

```
Matriz C:
```

```
30.0 36.0 42.0  
66.0 81.0 96.0  
102.0 126.0 150.0
```

Figura 123 Resultado en Python

SDK

```
-----Multiplicacion de Matrices---  
C=AB  
Matriz A  
A 1:  
A 2:  
A 3:  
A 4:  
A 5:  
A 6:  
A 7:  
A 8:  
A 9:  
Matriz B  
B 1:  
B 2:  
B 3:  
B 4:  
B 5:  
B 6:  
B 7:  
B 8:  
B 9:  
Realizado en 0.000004 segundos  
MAtriz A:  
1.500000 1.000000 2.500000  
1.000000 6.800000 3.000000  
4.000000 5.000000 6.000000  
MAtriz B:  
4.500000 3.500000 6.000000  
80.050003 6.000000 20.125000  
3.000000 4.000000 9.000000  
Matriz C (resultado):  
94.300003 21.250000 51.625000  
557.840027 56.300003 169.850006  
436.250000 68.000000 178.625000
```

Figura 124 Resultado obtenido en la terminal del SDK

Practica 10: ZYNQ con HDMI

Objetivo

- Desplegar video utilizando el puerto HDMI

Desarrollo

La práctica se divide en dos secciones:

- Vivado: Diseño del Sistema
- Creación de la aplicación:
 - SDK
 - Python

Vivado

Para el sistema se utilizarán IP's de la librería de Digilent, esta se puede acceder desde el siguiente enlace: <https://github.com/Digilent/vivado-library>, se pueden clonar los archivos o descargar el archivo zip.

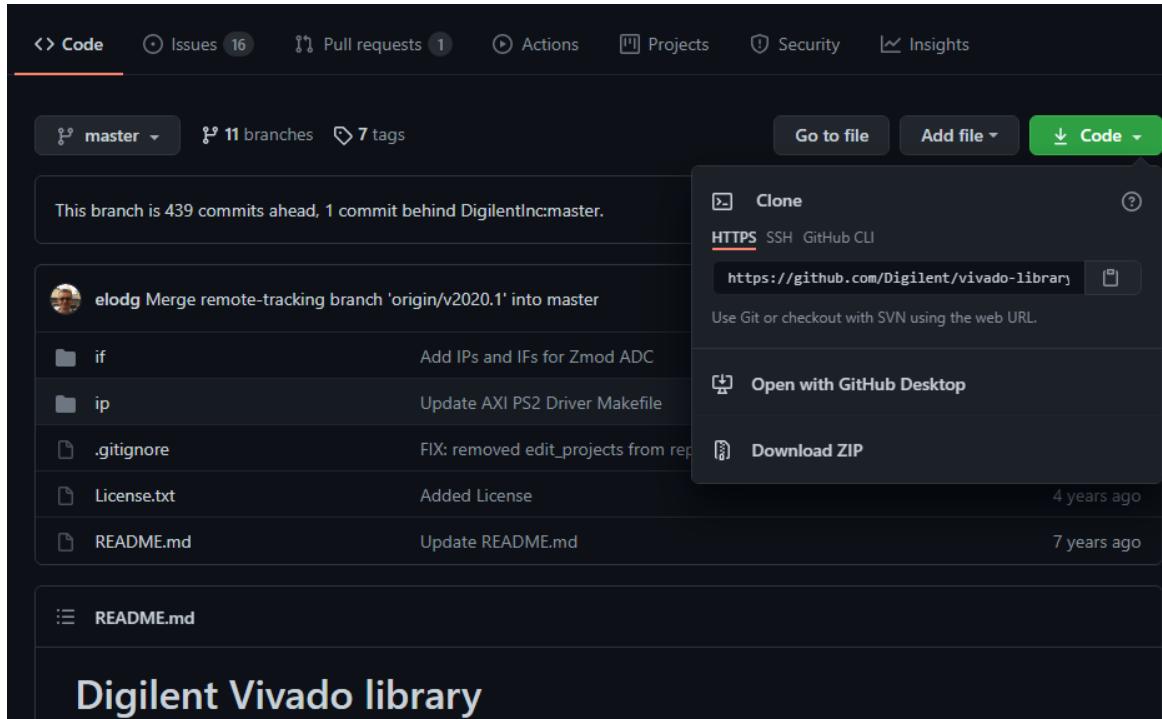


Figura 125 Libreria de Digilent

Es importante recordar la ruta en donde se clonaron los archivos o en donde se descomprimió el archivo zip. Ya que se tienen que añadir las IP's al proyecto de Vivado.

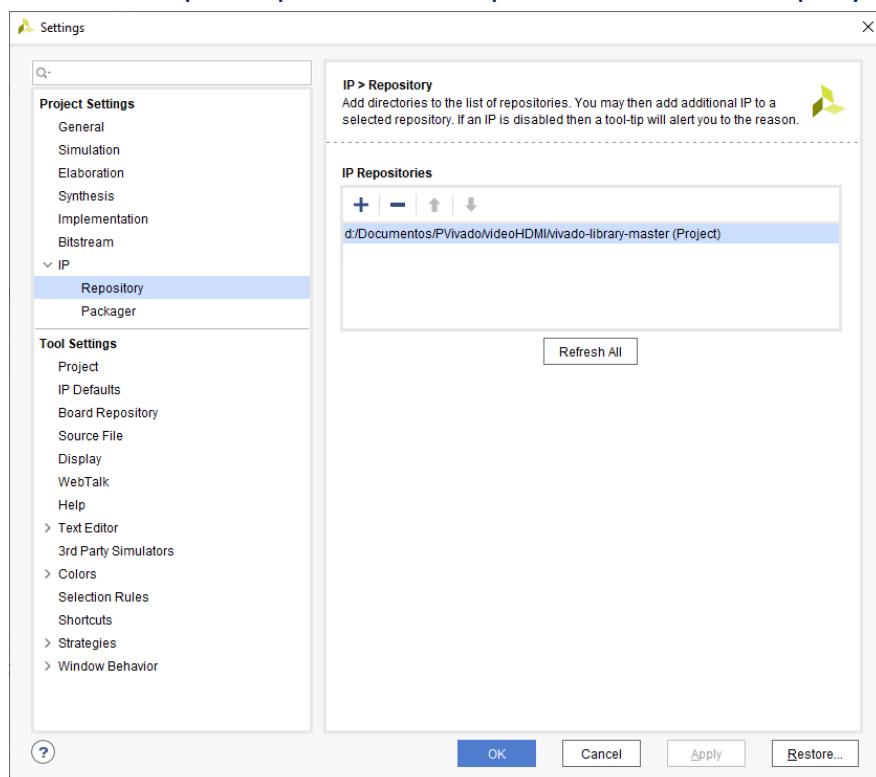


Figura 126 Configuración del repositorio de IP's.

Resultados

Anexo:Código C - GPIO

```
#include <stdio.h>
#include "platform.h"
#include "xgpio.h"
#include "xparameters.h"
#include "xil_printf.h" // la función menu se utiliza únicamente para desplegar las opciones
para controlar al mux
// cabe mencionar que el orden de las funciones puede ser arbitrario y tambien lo que se
pide
// para seleccionar cada operacion en este caso se utilziaron numeros,pueden ser letras
tambien
// pero se tendrian que hacer cambios en otras partes del codigo.
void menu(){
    print("Practica 1: ZYNQ + Modulo RTL \n\r");
    print("Elija la compuerta logica a realizar \n\r");
    print("    1 AND \n\r");
    print("    2 OR \n\r");
    print("    3 XOR \n\r");
    print("    4 NOR \n\r");
}
// la funcion validacion únicamente corrobora que lo ingresado sea una opcion existente
unsigned char validacion (){
    unsigned char A,AA; // variable que se obtendra del puerto serial (temporal y la que
se regresa)
    int u;// variable para la condición de while
    u=2;
    while(u!=1){
        A=XUartPs_RecvByte(XPAR_PS7_UART_0_BASEADDR);
        if ((A=='1')||(A=='2')||(A=='3')||(A=='4')){
            u=1;// como se ingreso una opcion valida se termina el ciclo haciendo
u==1
            AA=A;;
        }else{
            print("Ingrese una opcion valida \n\r");// mensaje para el usuario
            u=2;// se mantiene el ciclo while
        }
    }
    return AA;
}
// igual es una funcion para validar sin embargo aca únicamente se toma en cuenta cuando
// se ingresa el numero 5
unsigned char validacion2 (){
    unsigned char b,bb; // variable que se obtendra del puerto serial

    b=XUartPs_RecvByte(XPAR_PS7_UART_0_BASEADDR);
    if ((b=='5')||(b=' ')){
        bb=b;;
    }
    return bb;
}
//main es la función principal

int main()
```

```

{
    init_platform(); // inicializar la plataforma.
    int o=1; // variable que se utilizara en un ciclo while
    XGpio seleccion,salida; // creacion de las instancias seleccion y salida

    unsigned char sel,sel2; // variables donde se almacenaran las variables que retornan
    // de las funciones de validacion
    int resultado; // variable donde se guarda lo que se lee de la salida del mux
    // inicializacion de las entradas-salidas de propósito general
    // el GPIO_X debe coincidir con la numeracion del diagrama, es decir si la IP
    // axi_gpio_0 esta conectada con el selector del mux se le asigna la variable seleccion
    // para cambiar el nombre de la variable es en la sección XGpio linea 52
    XGpio_Initialize(&salida,XPAR_AXI_GPIO_0_DEVICE_ID);
    XGpio_Initialize(&seleccion,XPAR_AXI_GPIO_1_DEVICE_ID); //

    XGpio_SetDataDirection(&seleccion,1,0); // instancia - canal - registro de dirección
    de datos (1 para entradas / 0 para salidas)
    XGpio_SetDataDirection(&salida,1,1);

    print("ya paso la inicializacion\n\r");

    while(1)
    {
        menu(); // llamar a la función menu
        sel=validacion(); // llamar a la función validacion y guardarla en la variable sel
        o=1; // volver a crear el bucle de la función while(o==1)
        // lo que se escribe en cada opción depende del código en vhdl del mux
        switch (sel){
            case '1':
                XGpio_DiscreteWrite(&seleccion,1,0); // instancia - canal - dato
                break;
            case '2':
                XGpio_DiscreteWrite(&seleccion,1,1);
                break;
            case '3':
                XGpio_DiscreteWrite(&seleccion,1,2);
                break;
            case '4':
                XGpio_DiscreteWrite(&seleccion,1,3);
                break;
        }
        print("5 para regresar al menu \n\r");
        print("Ingrese cualquier carácter para que se imprima el resultado \n\r");

        while(o==1) // ciclo para imprimir el resultado
        {

            resultado=XGpio_DiscreteRead(&salida,1); // leer la salida y guardarla en
            resultado
            xil_printf("Resultado : %d \n\r",resultado); // imprimir el resultado
            sel2=validacion2(); // llamar la segunda validacion
            if (sel2=='5')
            {
                o=2; // romper el ciclo while
            }
        }

        cleanup_platform();
    }
}

```

```
    return 0;  
}
```