

# **PROCOM**

## **MicroBlaze**

Ariel Pola - Santiago Leguizamón  
[arielpola@gmail.com](mailto:arielpola@gmail.com) - [stgoleguizamon@gmail.com](mailto:stgoleguizamon@gmail.com)

September 2, 2023



# Tabla de Contenidos

- 1 Introducción
- 2 Nuevo Proyecto
- 3 Creando un nuevo diseño
- 4 Instancias de periféricos
- 5 Instancia de VIO
- 6 Instanciar el uP en Top Level
- 7 Instanciar el VIO en Top Level
- 8 Crear la aplicación en Vitis
- 9 Programar la FPGA y Ejecutar la Aplicación en Forma Local
- 10 Túnel, Programar la FPGA y Ejecutar la Aplicación en Forma Remota

# Tabla de Contenidos

## Sección 1

### Introducción

# Introducción

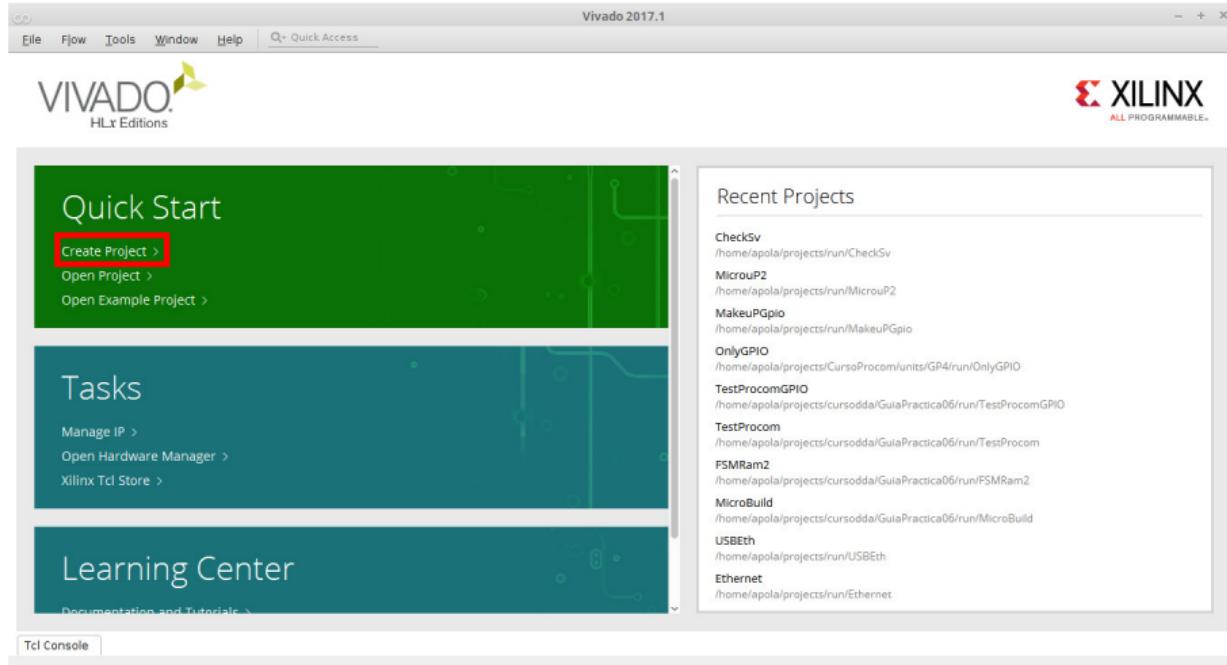
- El objetivo de esta presentación es detallar paso a paso la instancia y programación de un micro-embebido.
- El proyecto finaliza con el encendido de leds utilizando un script de python ejecutado en la PC y comunicándose con la FPGA utilizando el puerto UART y GPIOs hacia los leds.

# Tabla de Contenidos

## Sección 2

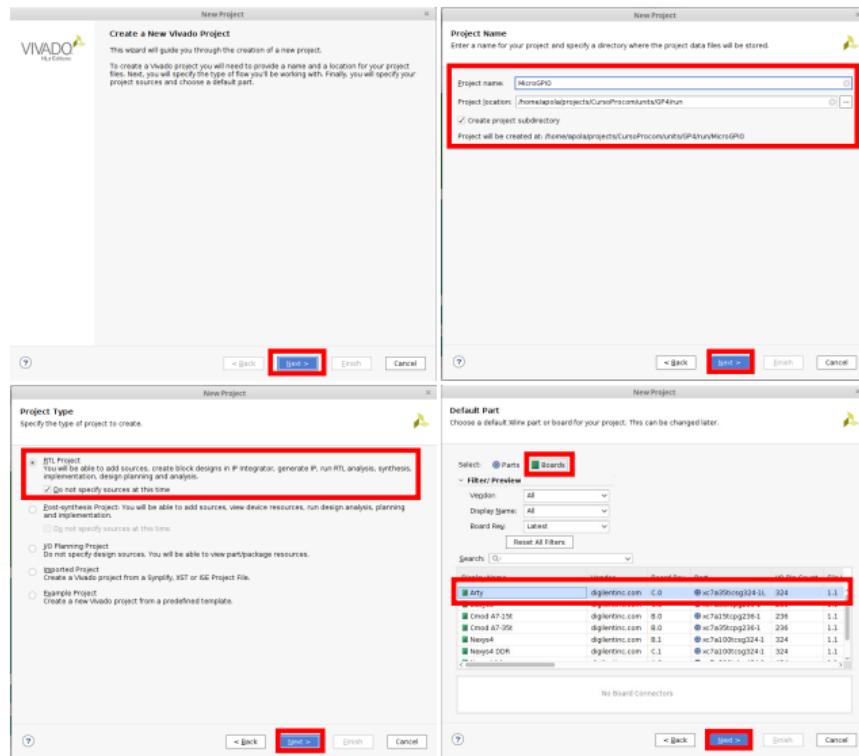
### Nuevo Proyecto

# Nuevo Proyecto



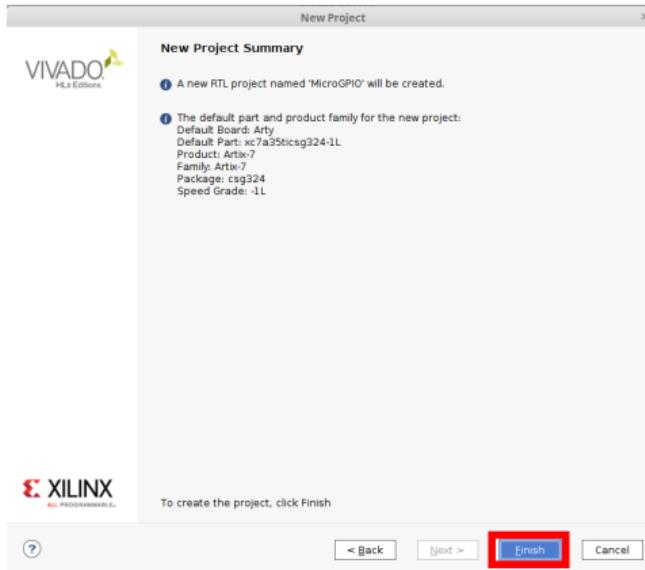
*Crear un nuevo proyecto.*

# Nuevo Proyecto



*Definir un nombre del proyecto, directorio de trabajo y kit Arty.*

# Nuevo Proyecto



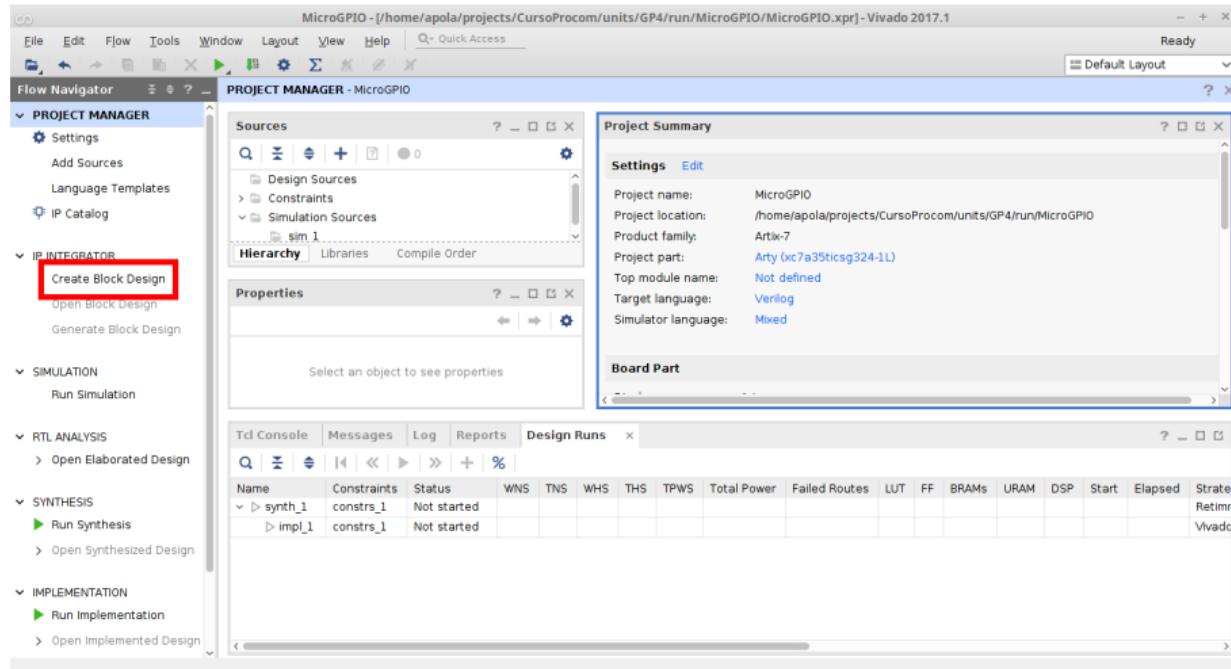
*Resumen del proyecto.*

# Tabla de Contenidos

## Sección 3

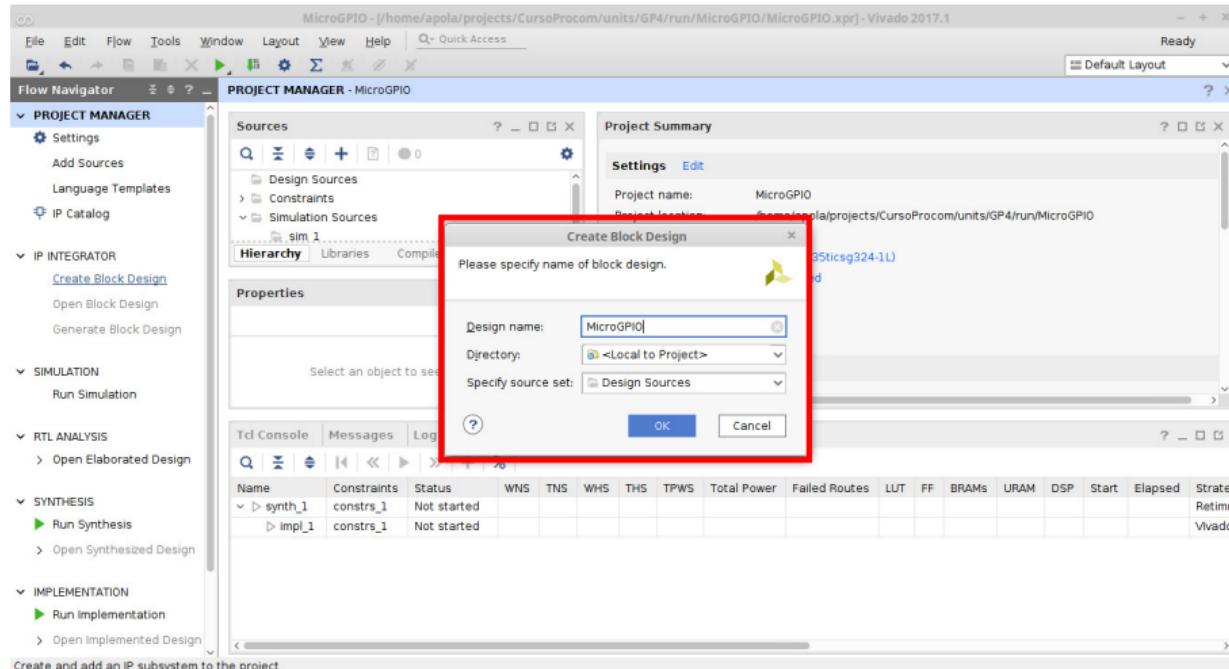
### Creando un nuevo diseño

# Creando un nuevo diseño



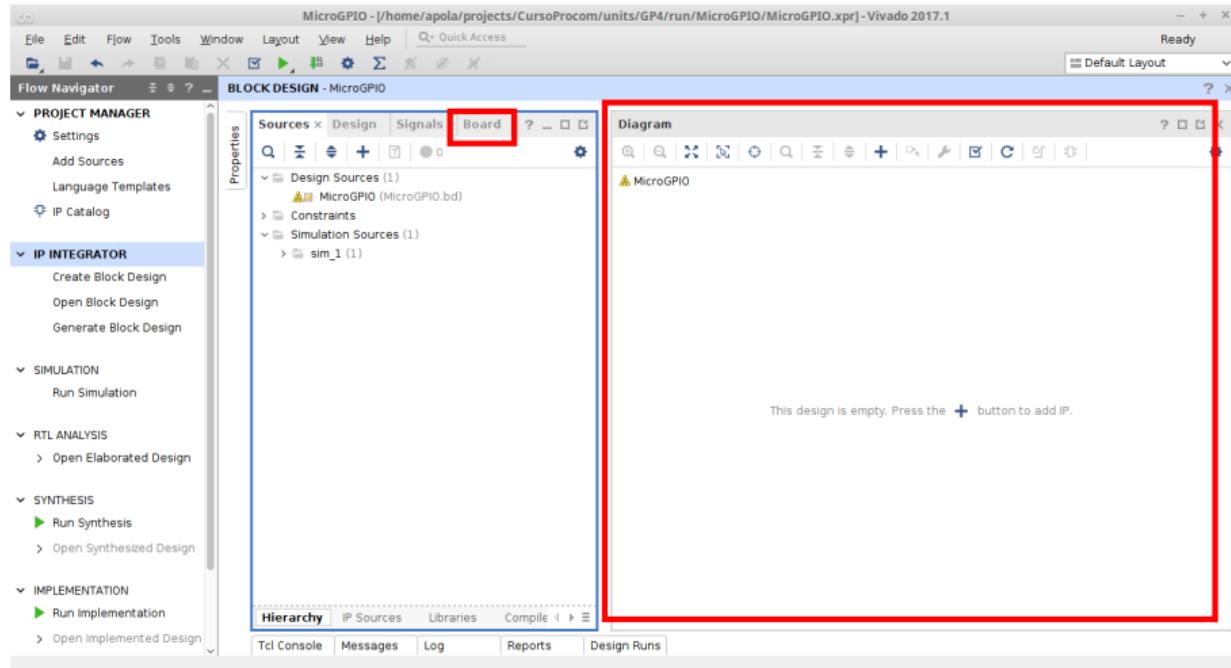
*Crean un nuevo bloque.*

# Creando un nuevo diseño



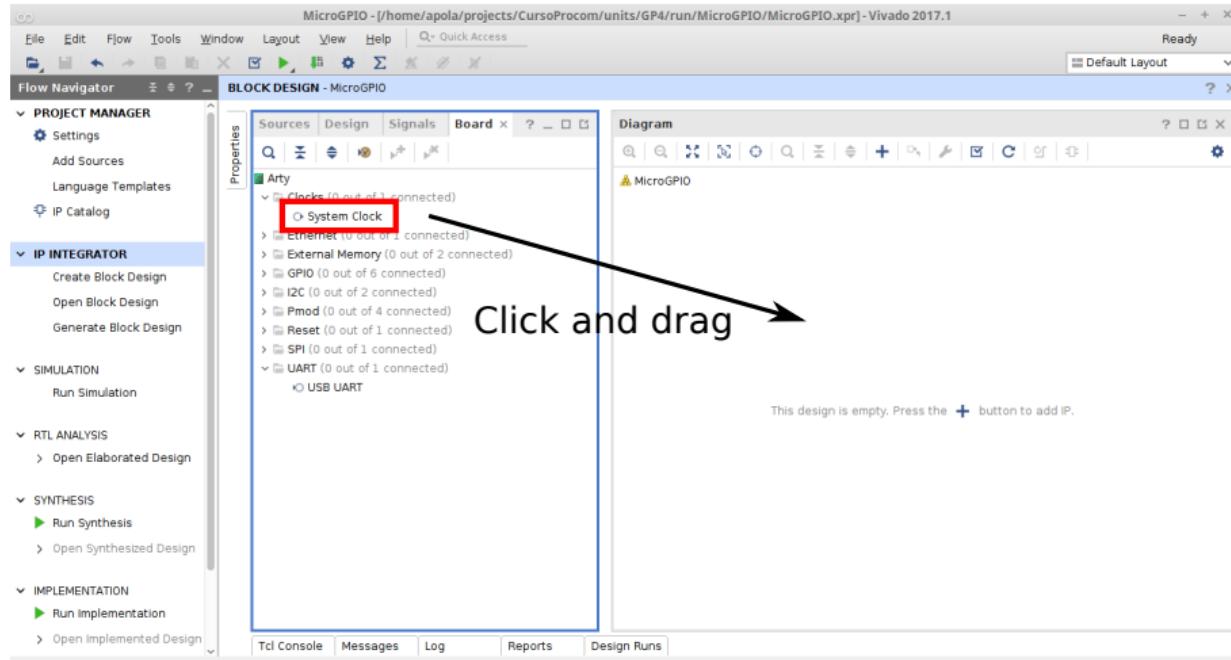
Definir el nombre del nuevo diseño y el directorio de trabajo.

# Creando un nuevo diseño



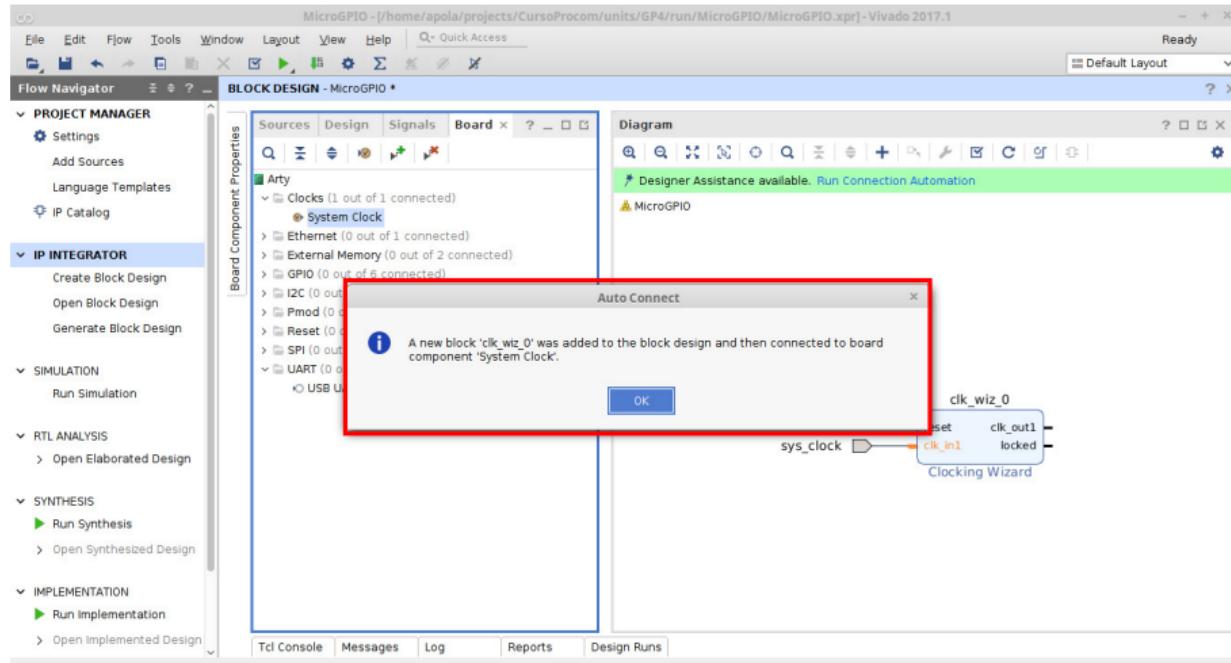
Nueva ventana de trabajo denominada “Diagrama” y en la pestaña “Board” se definen todos los componentes de la FPGA seleccionada.

# Creando un nuevo diseño



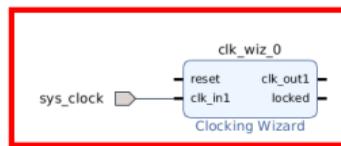
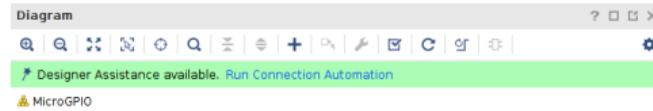
*Instanciar "System Clock" haciendo click y arrastrando hacia la ventana.*

# Creando un nuevo diseño



*Mensaje de instancia de bloque.*

# Creando un nuevo diseño

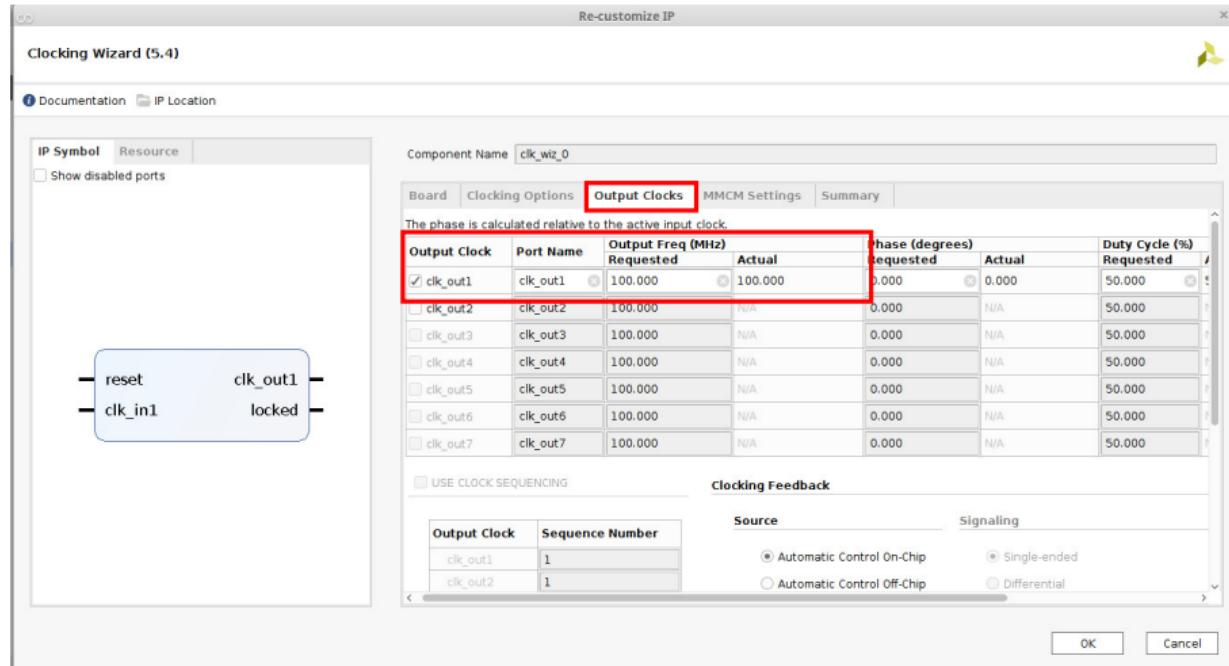


Double Click

---

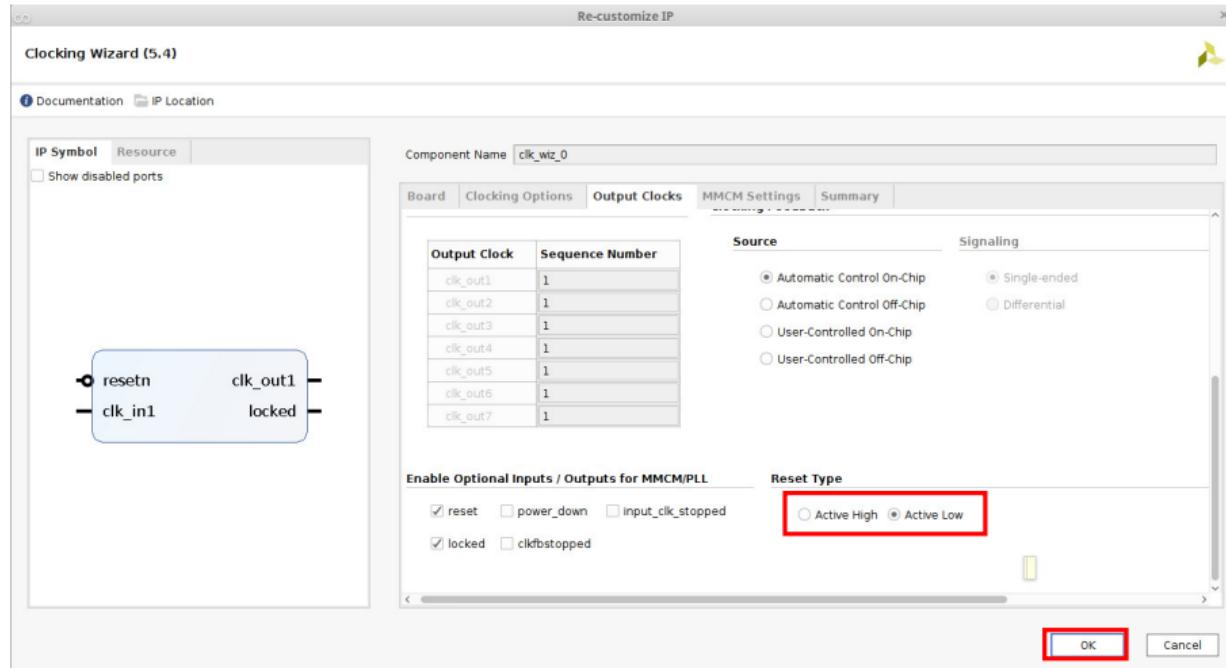
*Hacer doble click sobre “Clocking Wizard”.*

# Creando un nuevo diseño



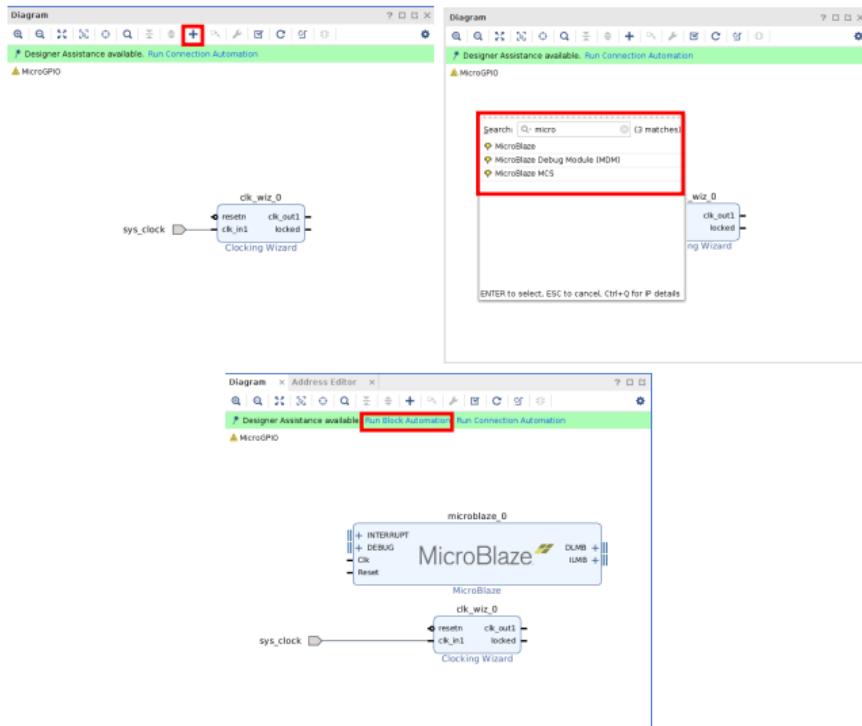
En la pestaña “Output Clocks” asignar la frecuencia de salida en 100MHz.

# Creando un nuevo diseño



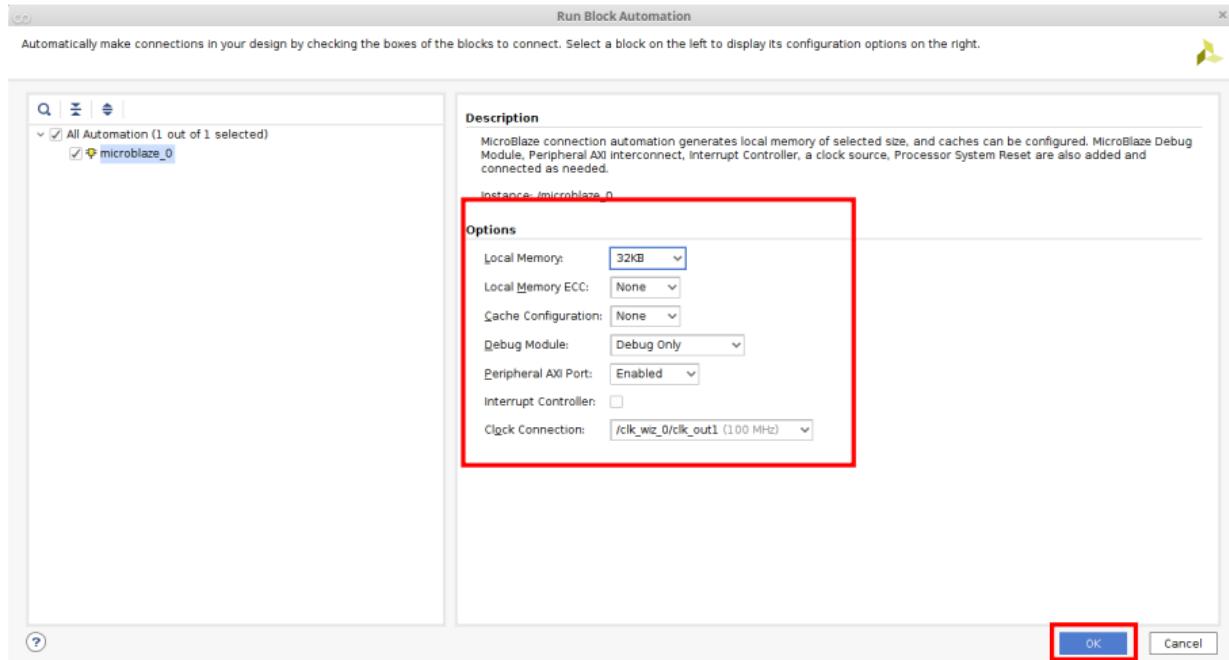
*El reset del bloque tiene que ser activo por bajo.*

# Creando un nuevo diseño



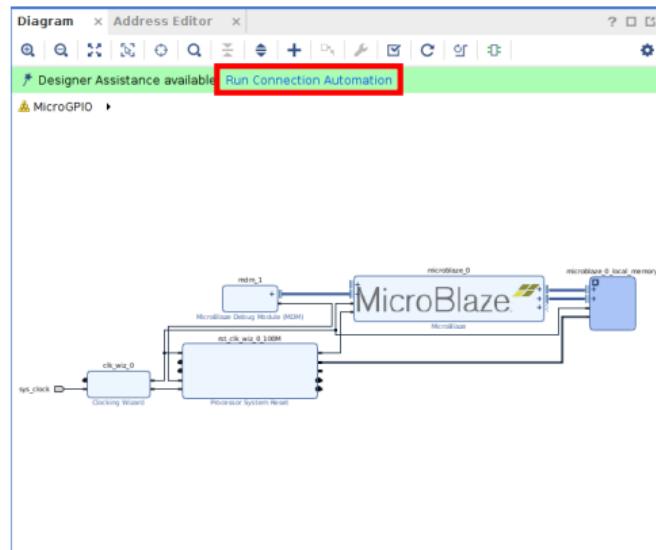
Incluir el core del MicroBlaze y ejecutar "Run Block Automation".

# Creando un nuevo diseño



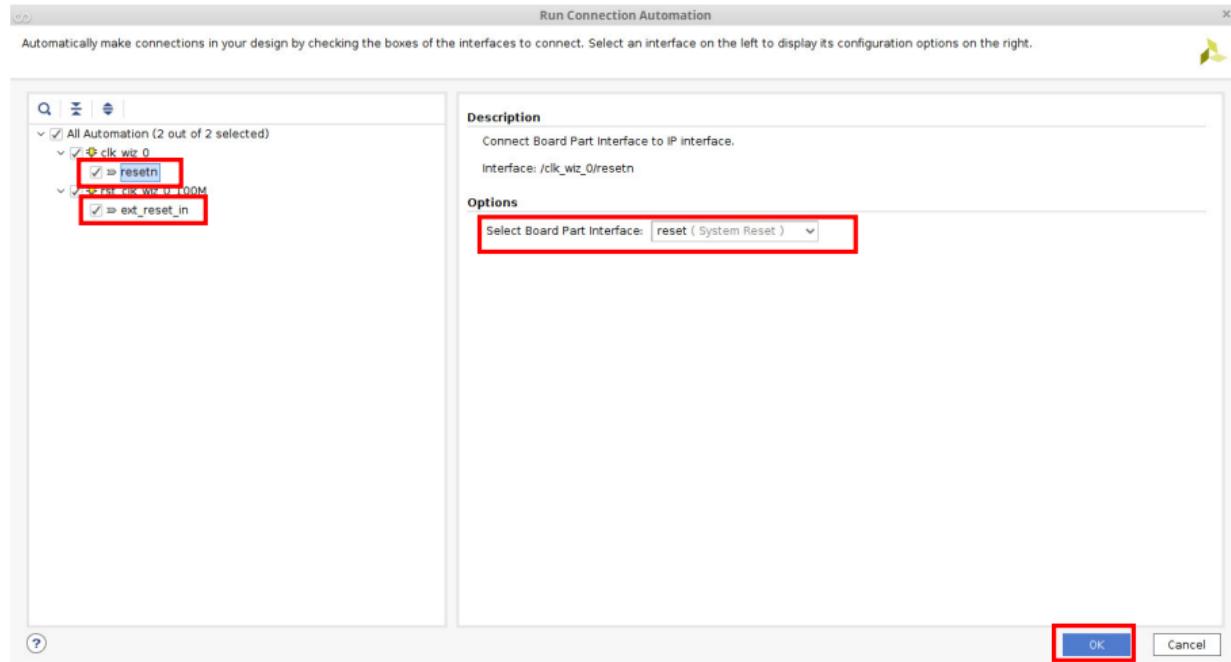
*Signar los parámetros de diseño del micro.*

# Creando un nuevo diseño



*Auto-conectar los elementos utilizando “Run Connection Automation”.*

# Creando un nuevo diseño

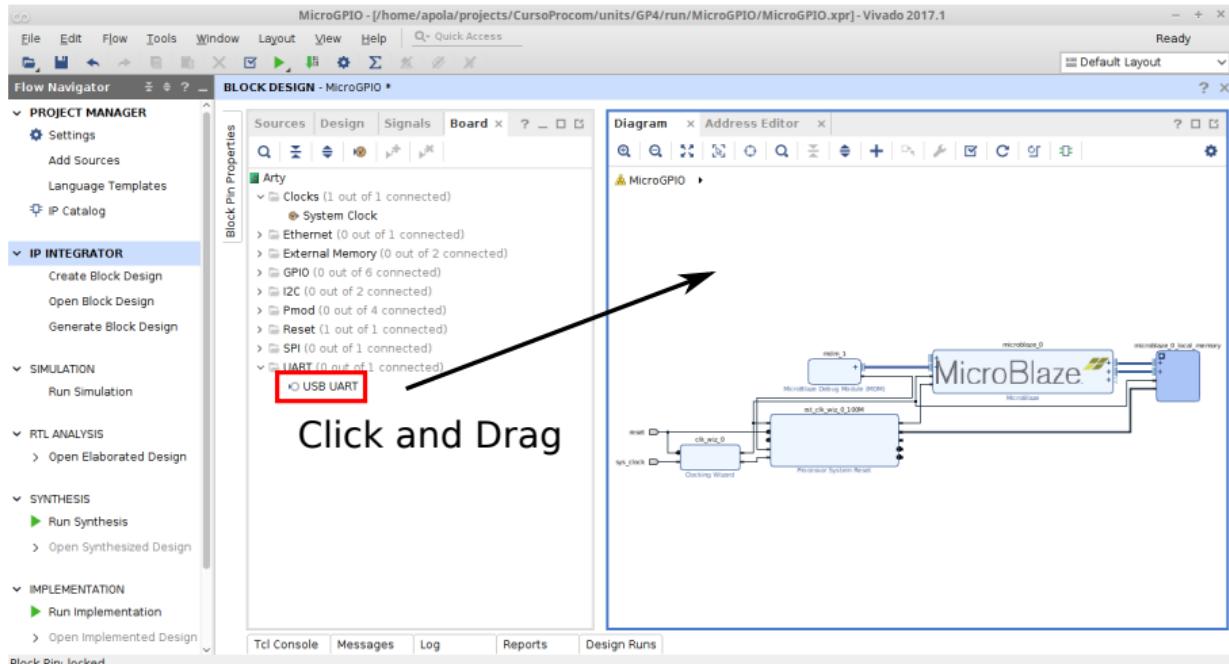


*Verificar que los dos puertos de reset estén conectados al reset del sistema.*

## Sección 4

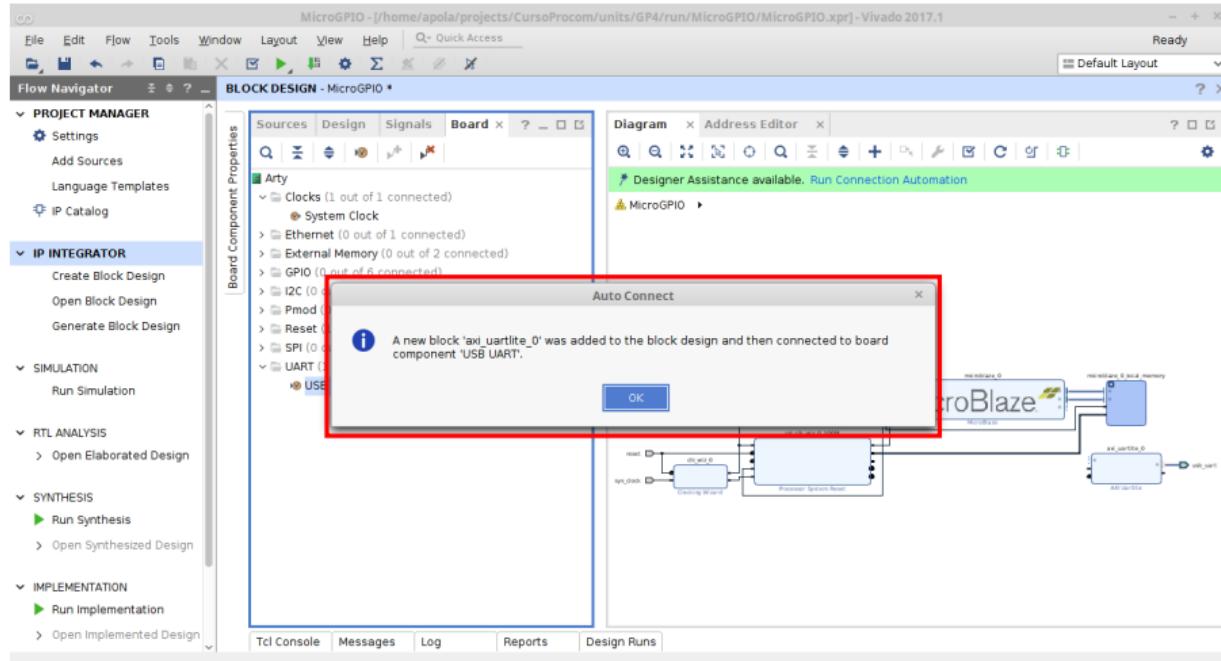
### Instancias de periféricos

# Instancias de periféricos



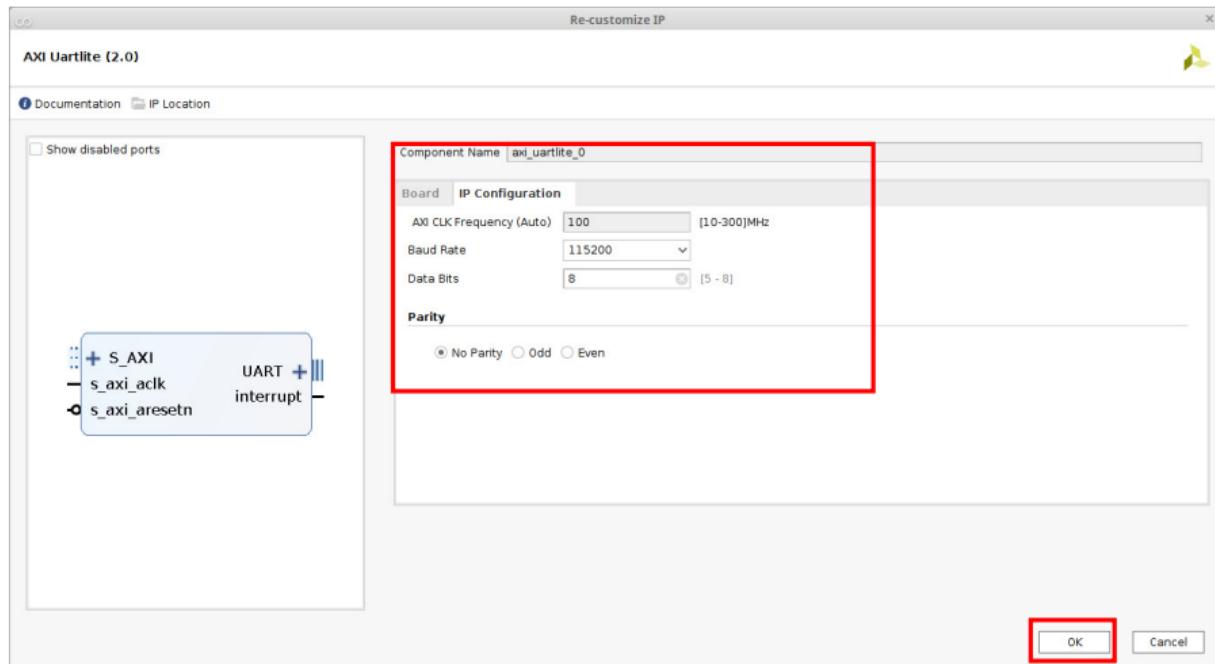
*Instanciar el IP USB UART.*

# Instancias de periféricos



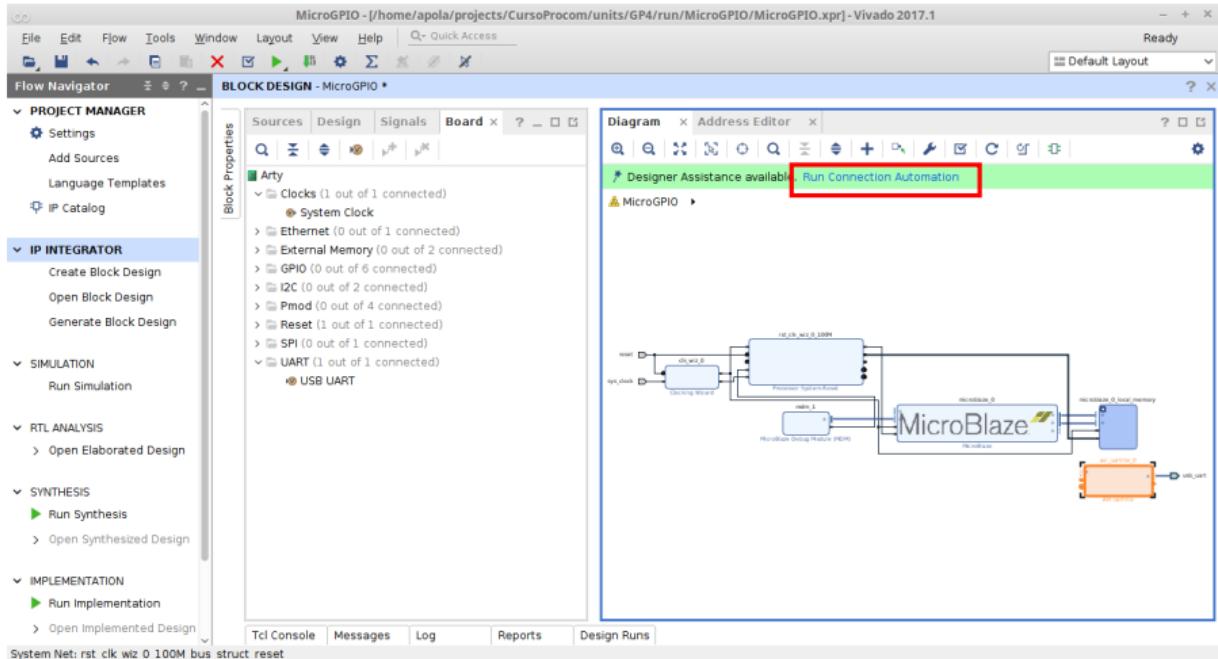
Verificar la instancia.

# Instancias de periféricos



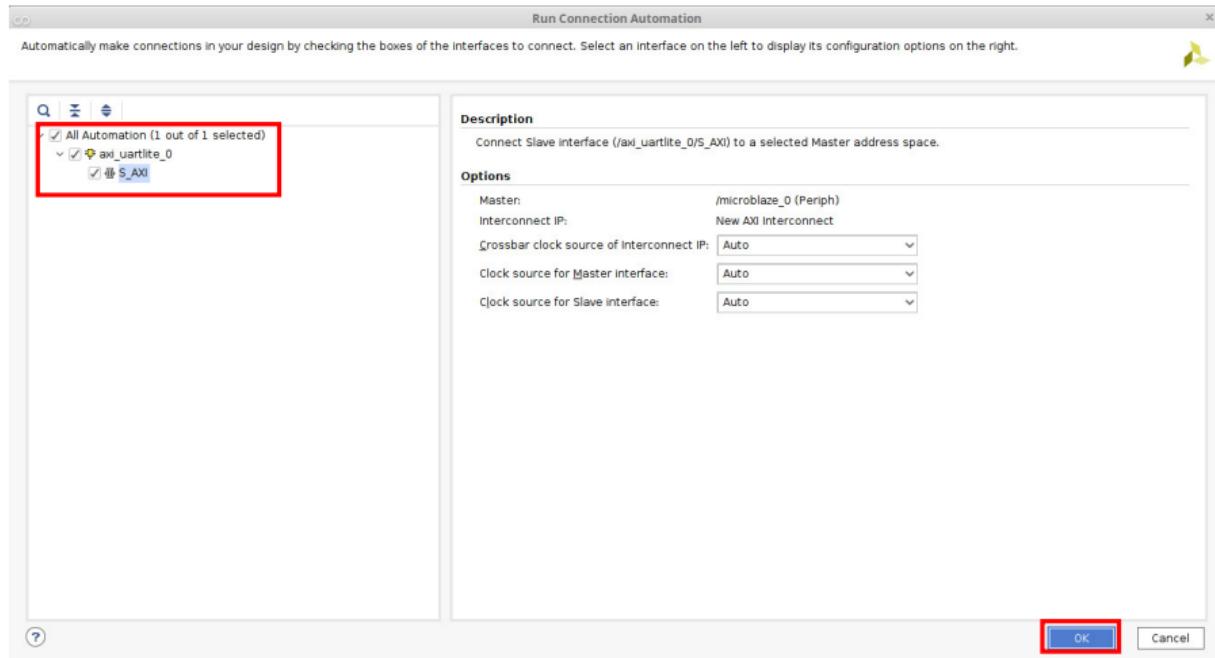
Hacer doble click sobre el periférico *UART* y configurar el *Baud Rate* en 115200.

# Instancias de periféricos



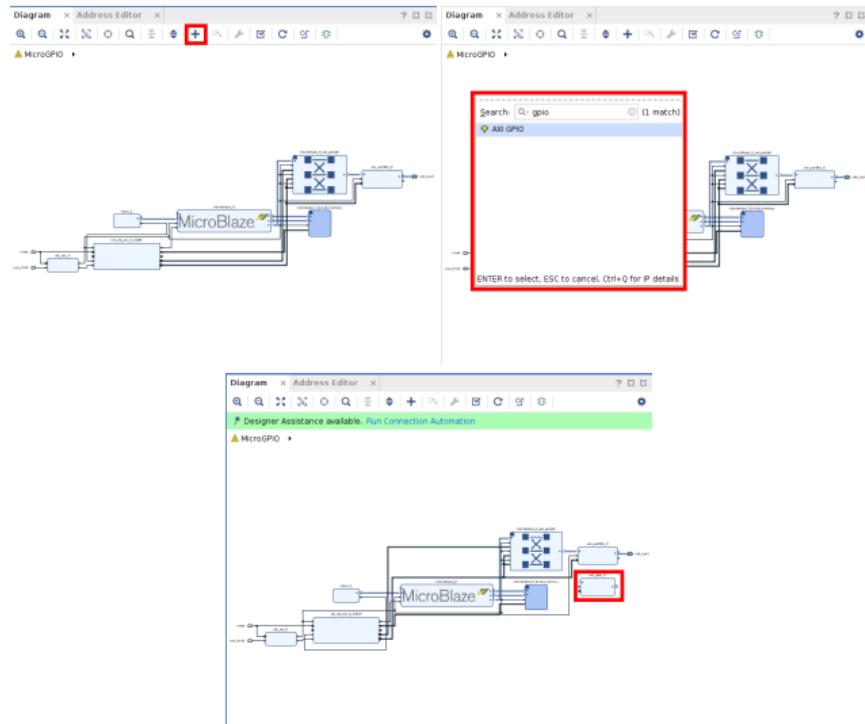
Seleccionar la conexión automática.

# Instancias de periféricos



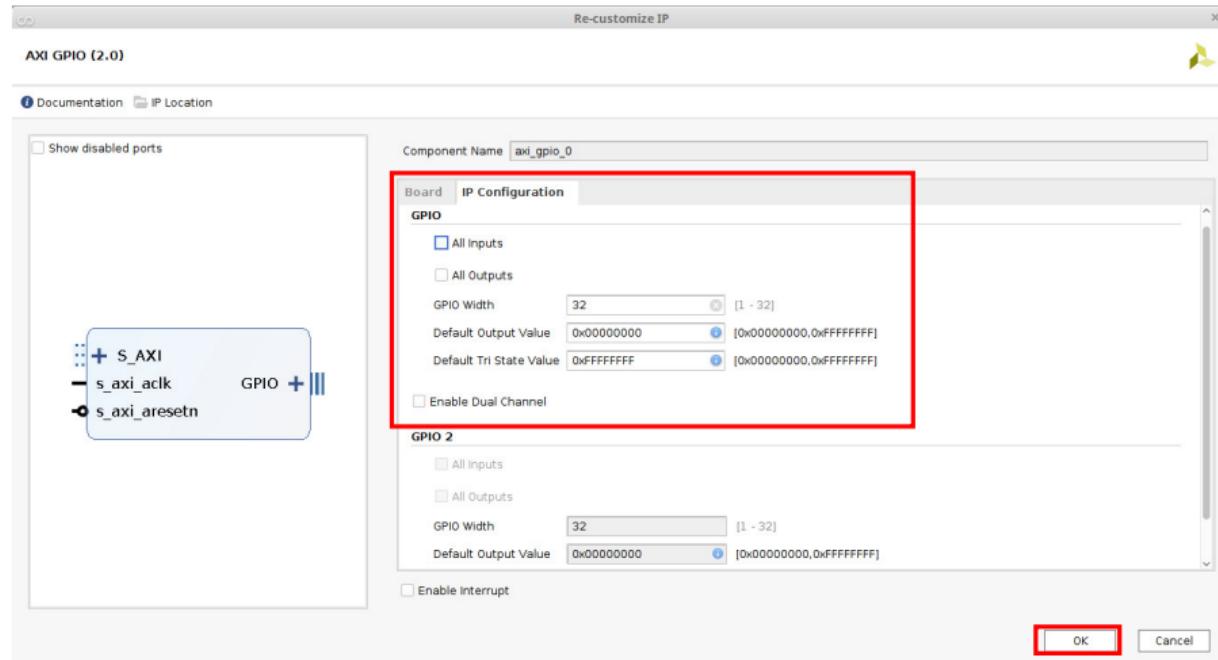
Verificar que todos los puertos estén seleccionados.

# Instancias de periféricos



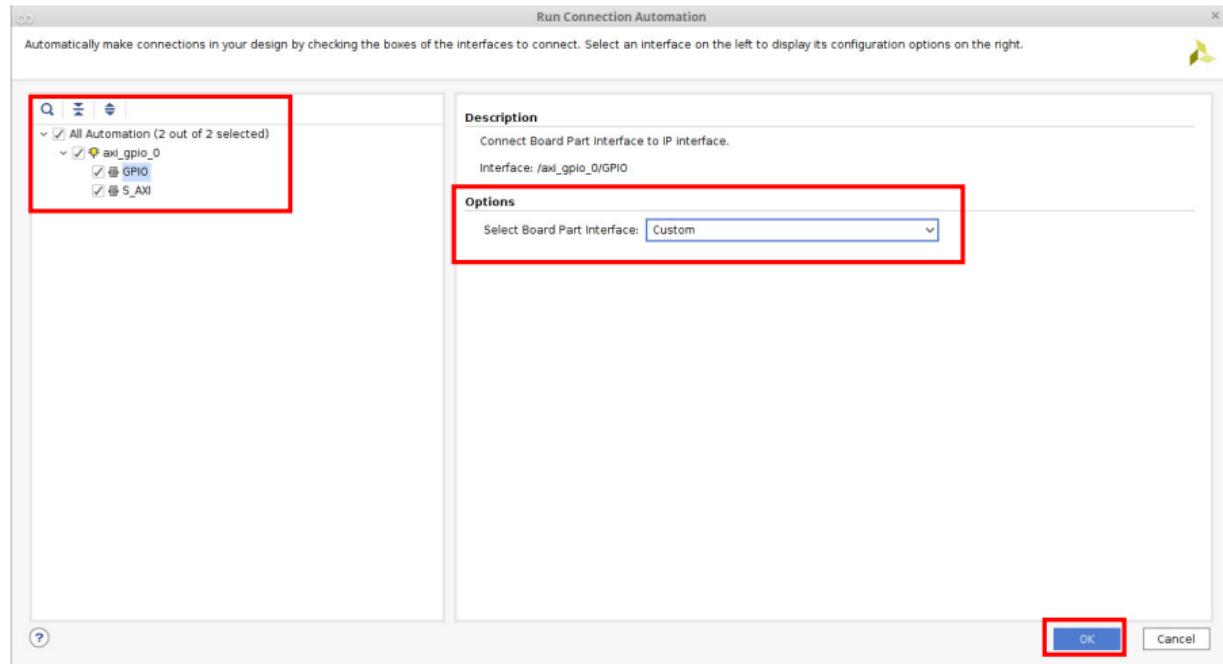
*Agregar un nuevo componente GPIO y hacer doble click sobre el IP.*

# Instancias de periféricos



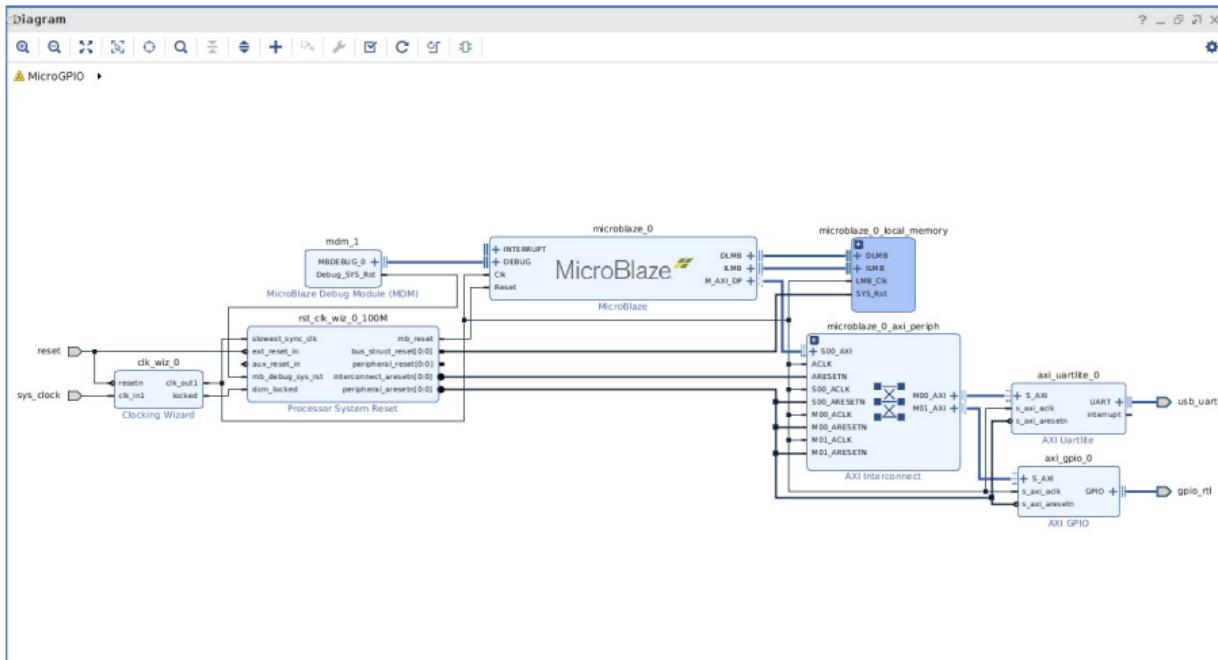
Verificar la configuración del puerto.

# Instancias de periféricos



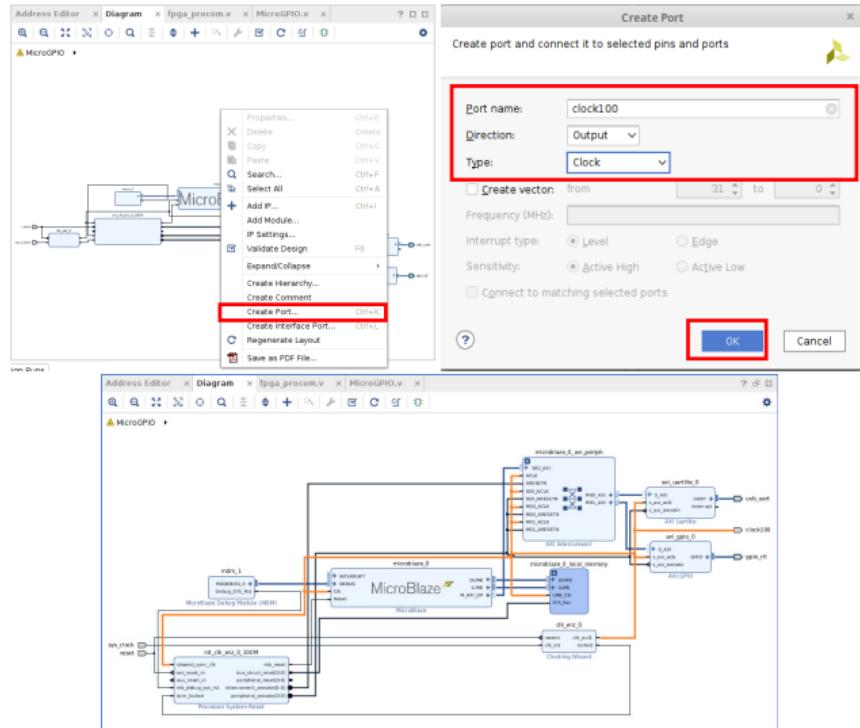
Seleccionar "Run Connection Automation" y verificar que el puerto de salida sea "Custom".

# Instancias de periféricos



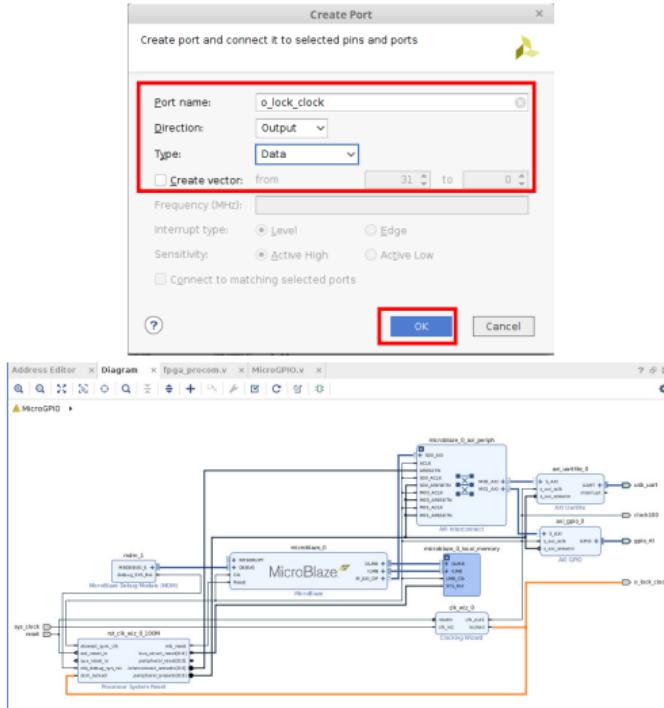
Diseño preliminar.

## Instancias de periféricos



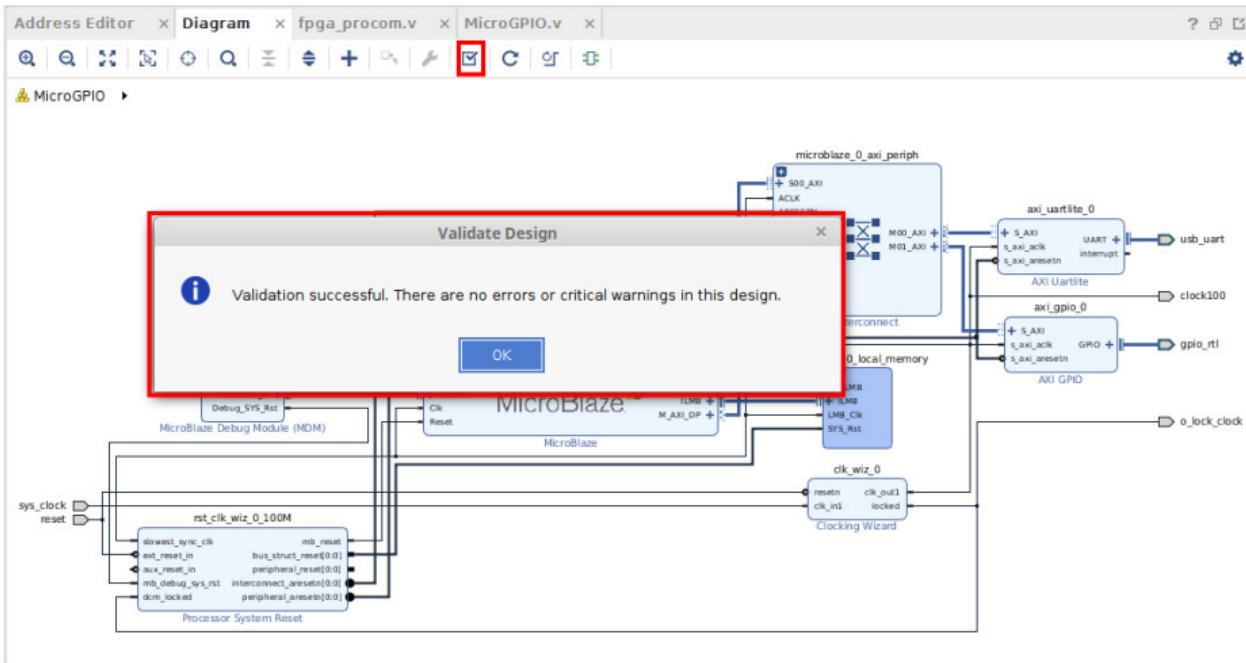
*Incluir un nuevo puerto de clock de salida.*

# Instancias de periféricos



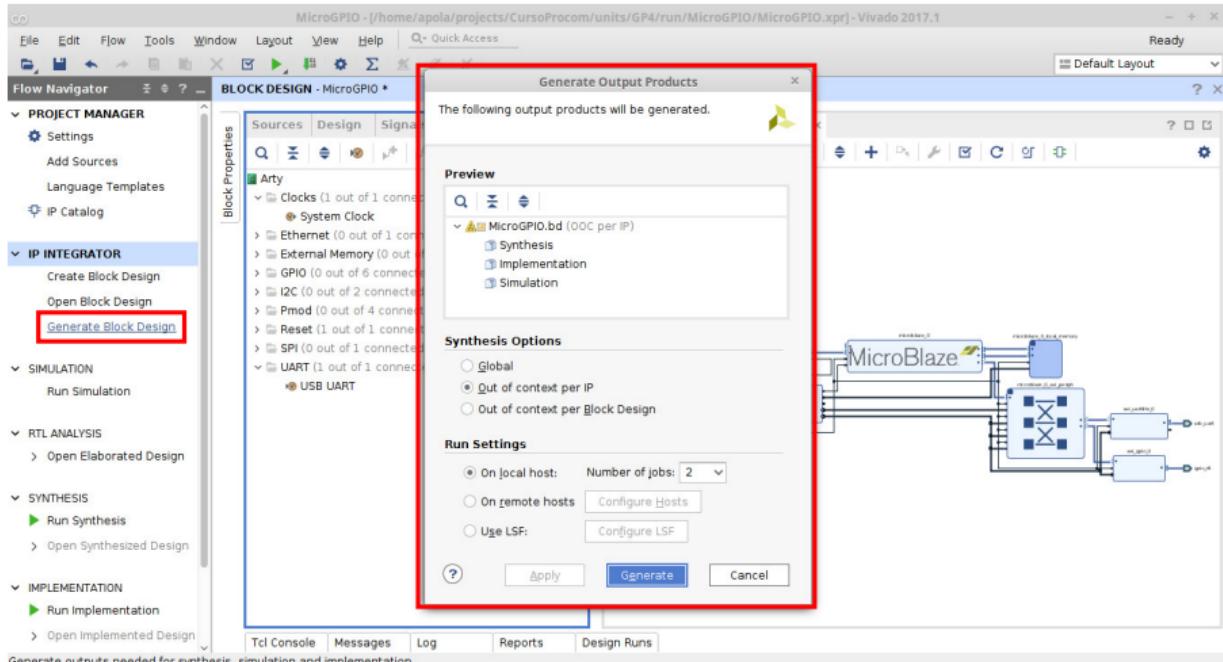
*Repetir el paso anterior incluyendo un nuevo puerto de lock de clock.*

## Instancias de periféricos



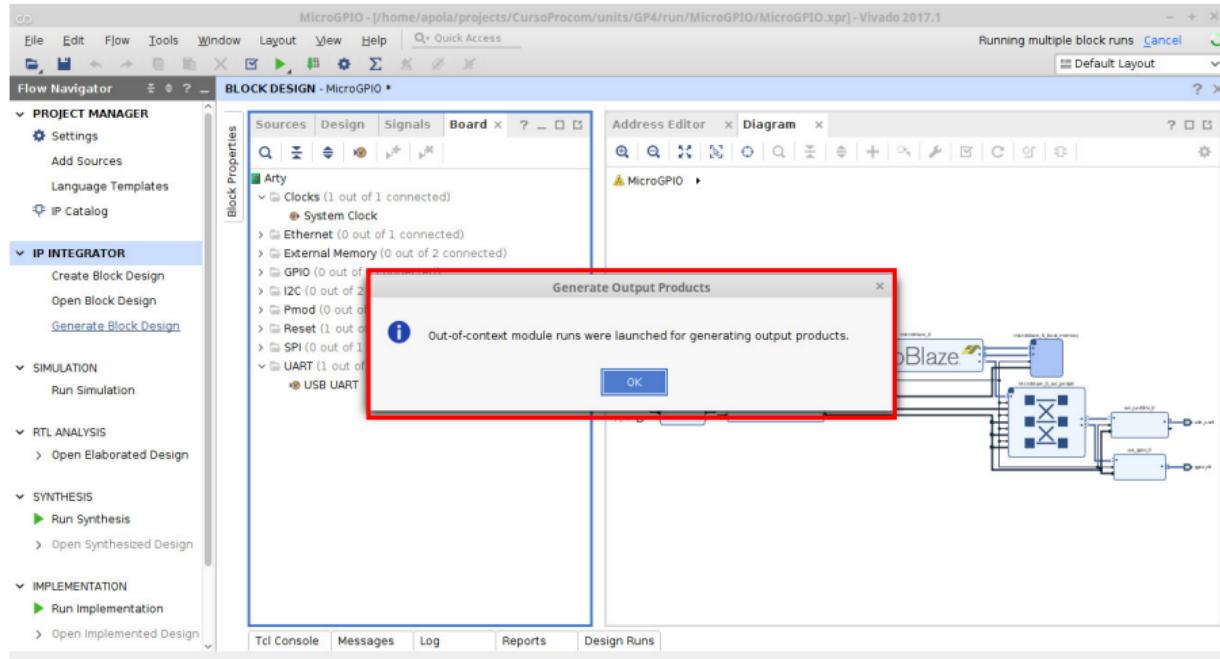
## *Validar el diseño.*

# Instancias de periféricos



Generar el diseño completo.

# Instancias de periféricos



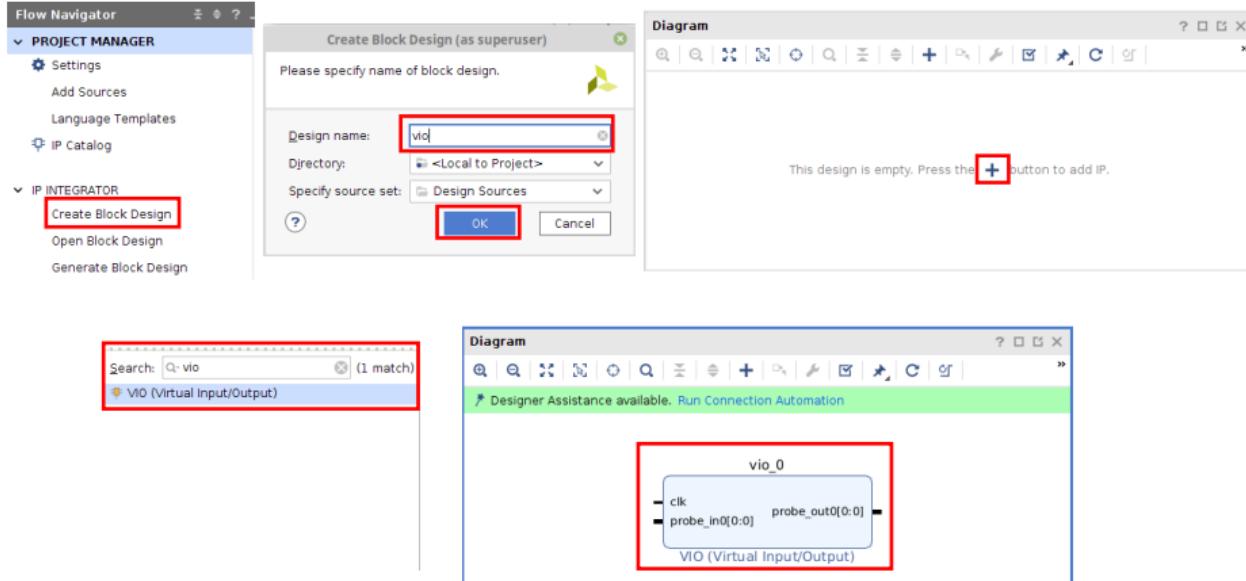
Verificar la elaboración.

# Tabla de Contenidos

## Sección 5

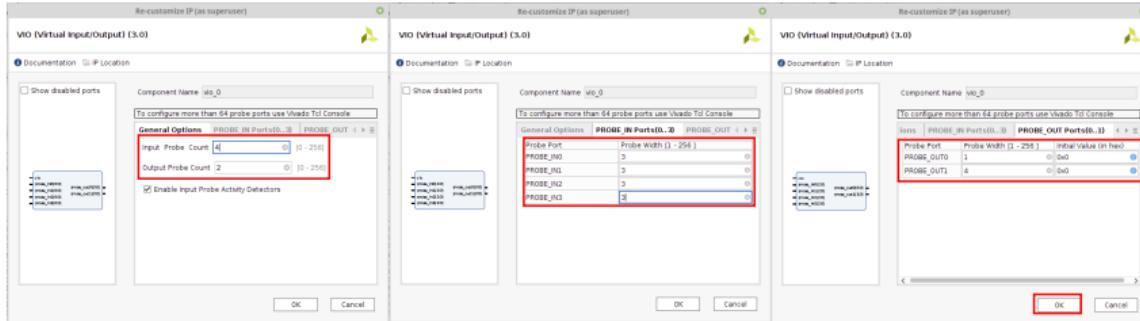
### Instancia de VIO

# Instancia de VIO



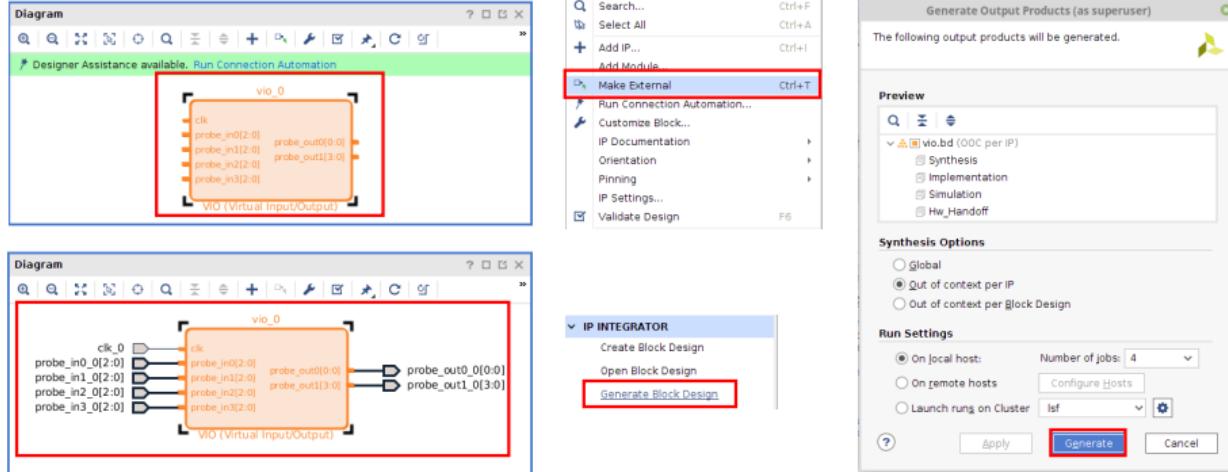
En caso de necesitar controlar en forma remota la FPGA debemos incluir el bloque VIO a nuestro diseño. Creamos un nuevo bloque, le asignamos el nombre, agregamos un nuevo IP, seleccionamos el IP VIO y hacemos doble click sobre el IP generado.

# Instancia de VIO



En el ejemplo vamos a habilitar el control externo, controlar los SWITCH y observar el valor de los leds RGB. Agregamos 4 puertos de entrada y 2 de salida. Cada puerto de entrada tiene 3 bits. En el caso de los puertos de salida se tienen 1 y 4 bits respectivamente.

# Instancia de VIO

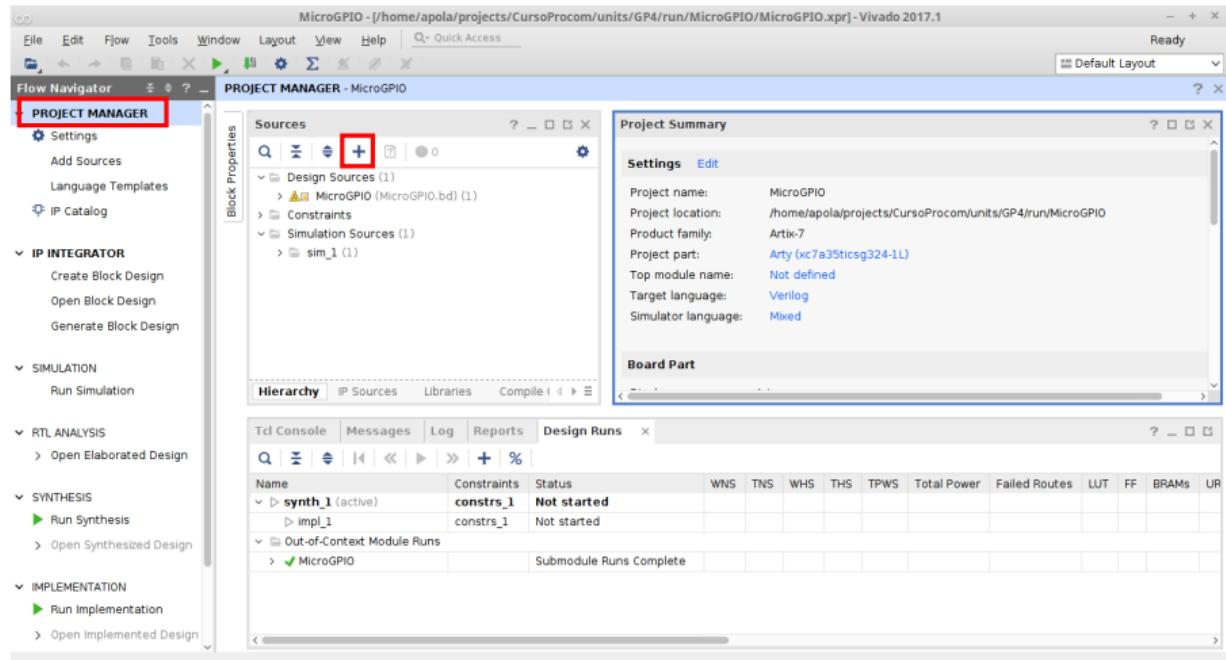


Sobre el bloque modificado hacemos click con el botón derecho, seleccionamos Make External lo cual agrega los puertos exteriores y por último generamos el bloque.

## Sección 6

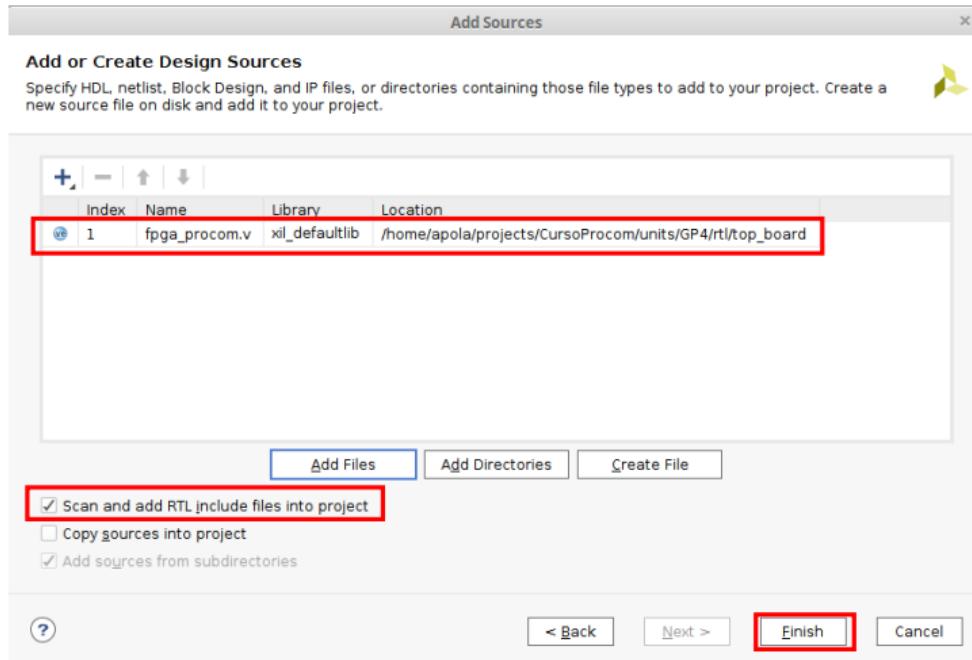
### Instanciar el uP en Top Level

# Instanciar el uP en Top Level



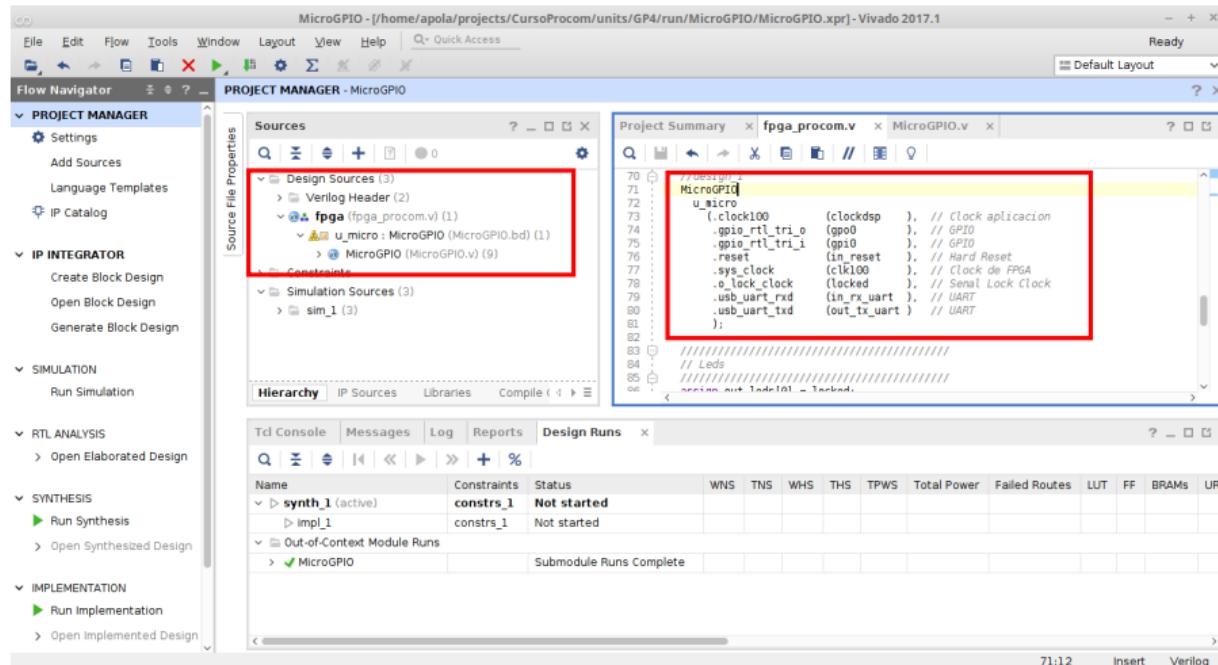
Agregar un nuevo archivo verilog.

# Instanciar el uP en Top Level



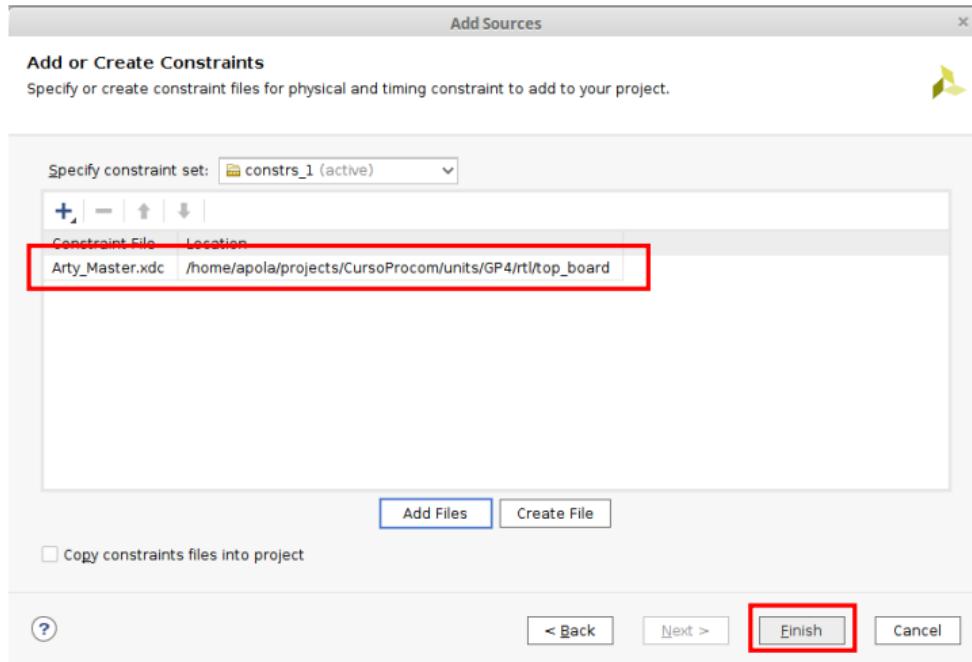
Seleccionar el archivo “fpga\_procom” y asegurarse de activar el escaneo de includes.

# Instanciar el uP en Top Level



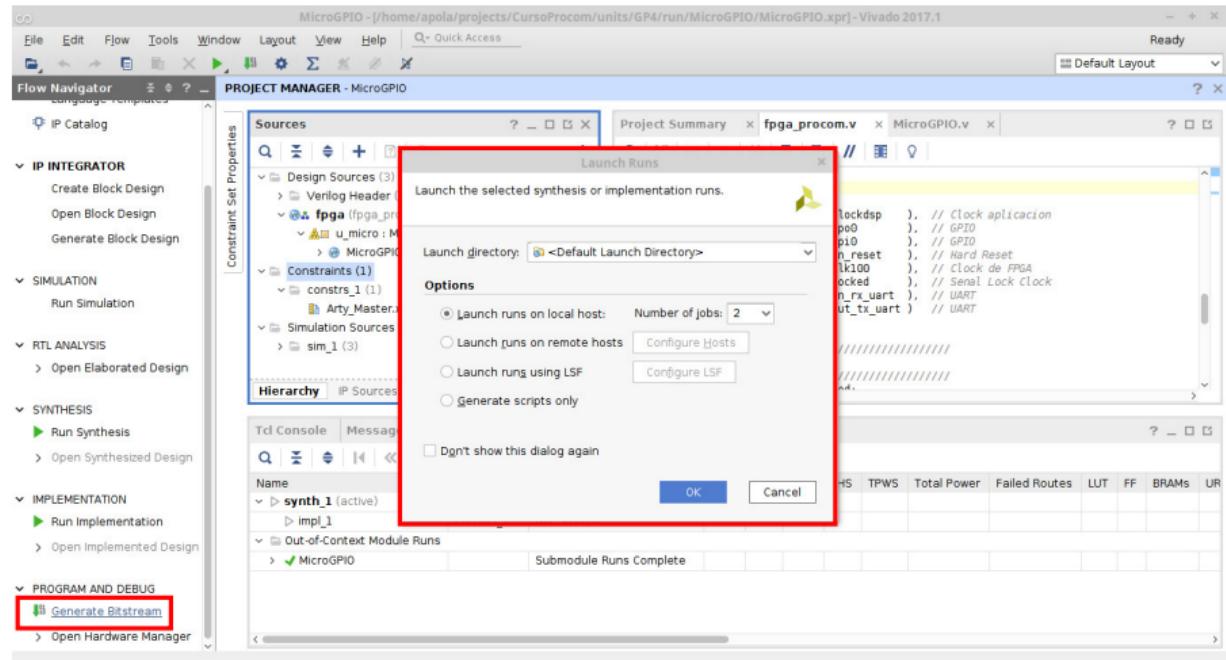
Verificar que el nombre del archivo del micro sea el mismo que se utiliza en el top level y los mismos puertos.

# Instanciar el uP en Top Level



*Agregar el archivo de puertos físicos.*

# Instanciar el uP en Top Level



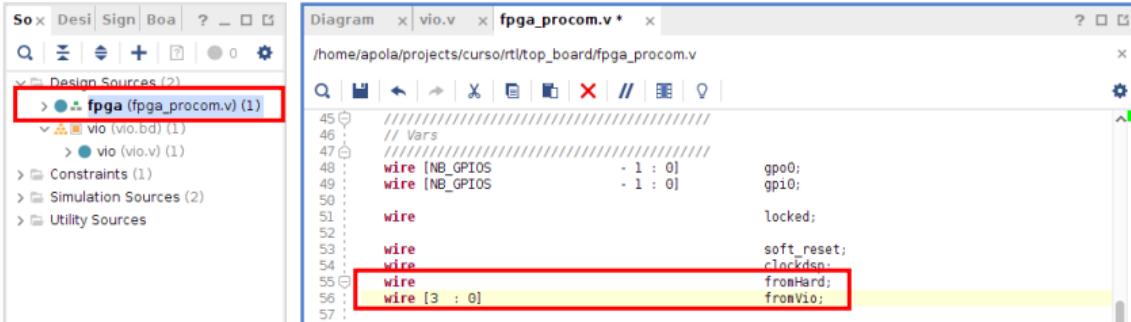
Generar el binario.

## Sección 7

### Instanciar el VIO en Top Level

# Instanciar el VIO en Top Level

- En caso que se esté trabajando en forma remota, vamos a necesitar incluir el VIO para controlar los switch y observar los valores de los puertos de salida.



```
Diagram x vio.v x fpga_procom.v * x
/home/apola/projects/curs.../top_board/fpga_procom.v
? □ □
Design Sources (2)
> fpga (fpga_procom.v) (1)
> vio (vio.bd) (1)
> vio (vio.v) (1)
Constraints (1)
Simulation Sources (2)
Utility Sources

Diagram x vio.v x fpga_procom.v * x
? □ □
Design Sources (2)
> fpga (fpga_procom.v) (1)
> vio (vio.bd) (1)
> vio (vio.v) (1)
Constraints (1)
Simulation Sources (2)
Utility Sources

45 //////////////////////////////////////////////////////////////////
46 // Vars
47 //
48 wire [NB_GPIOS] gpo0;
49 wire [NB_GPIOS] gpo1;
50
51 wire
52
53 wire
54
55 wire soft_reset;
56 wire clock;
57 wire fromHard;
58 wire fromVio;
```

Declarar en el módulo Top dos variables wire de 1 y 4 bits respectivamente. La primera nos permitirá seleccionar de donde queremos controlar el dispositivo y la segunda es para emular el comportamiento de los switch en forma virtual.

# Instanciar el VIO en Top Level

The screenshot shows the Vivado IDE interface with three main windows:

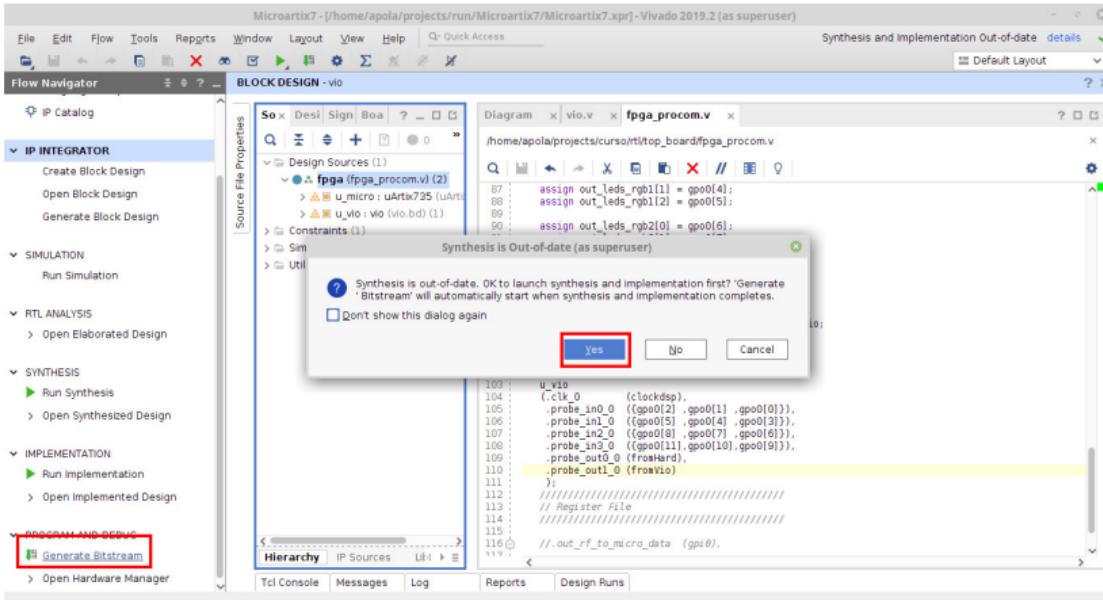
- Sources**: Shows the project structure with "fpga\_procom.v" and "vio(vio.bd)" under "Design Sources".
- Diagram**: Displays the Verilog code for "vio.v". A red box highlights the module declaration:

```
module vio
  (clk_0,
```
- Diagram**: Displays the Verilog code for "fpga\_procom.v". A red box highlights the instantiation of the VIO module:

```
u_vio
  .clk_0(clockdsp),
```

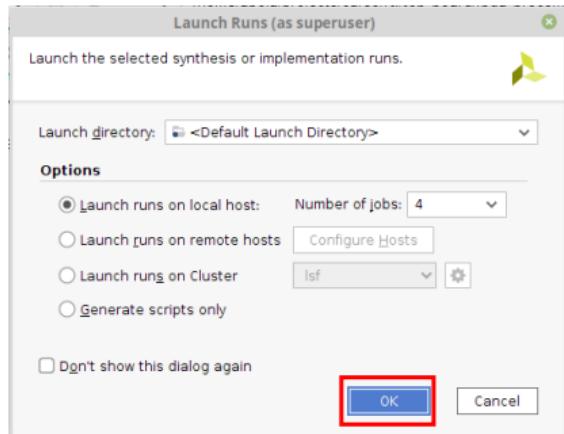
Además, debemos copiar la declaración del módulo VIO e incluirlo en el Top Level. Por último, asignamos la selección del control por medio del bloque VIO y comentamos la asignación directa de los switch.

# Instanciar el VIO en Top Level



Generamos el binario nuevamente.

# Instanciar el VIO en Top Level

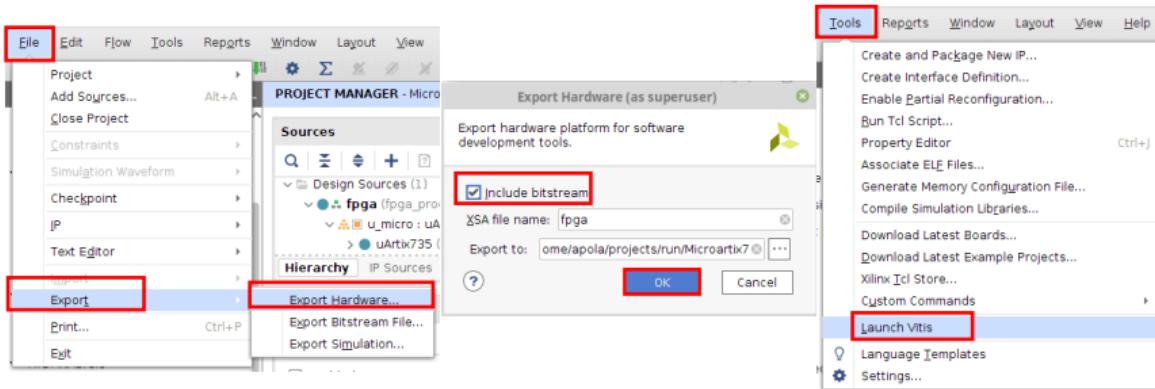


*Configuramos cuantos cores queremos utilizar para generar el binario.*

## Sección 8

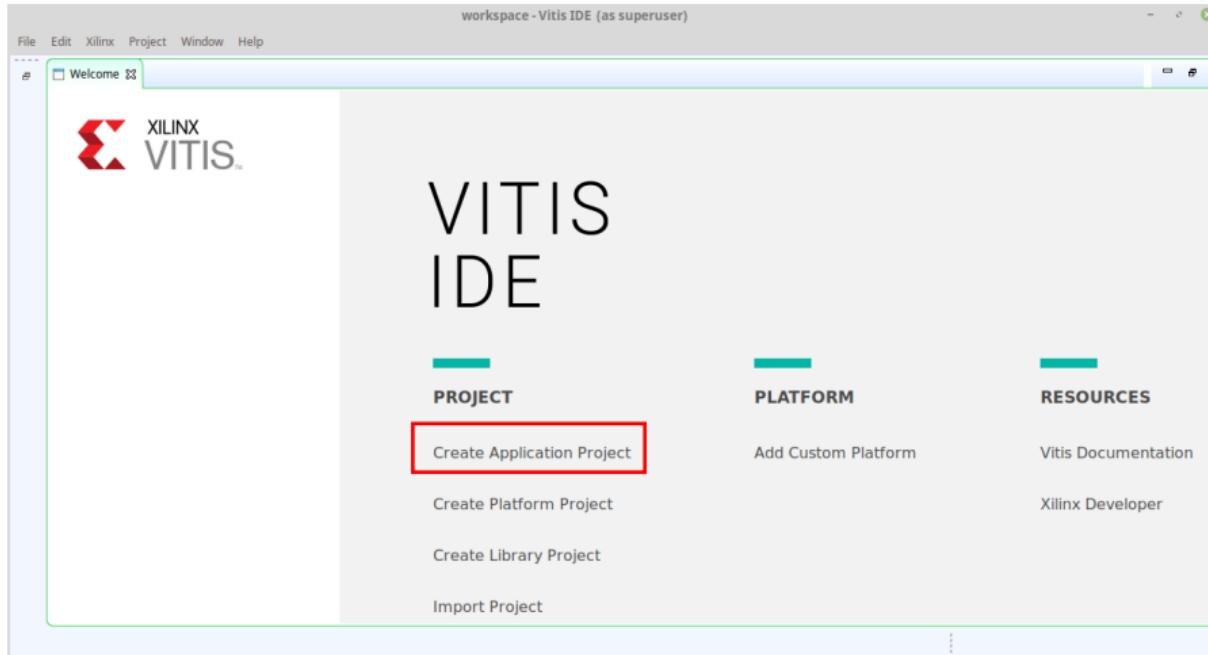
Crear la aplicación en Vitis

# Crear la aplicación en Vitis



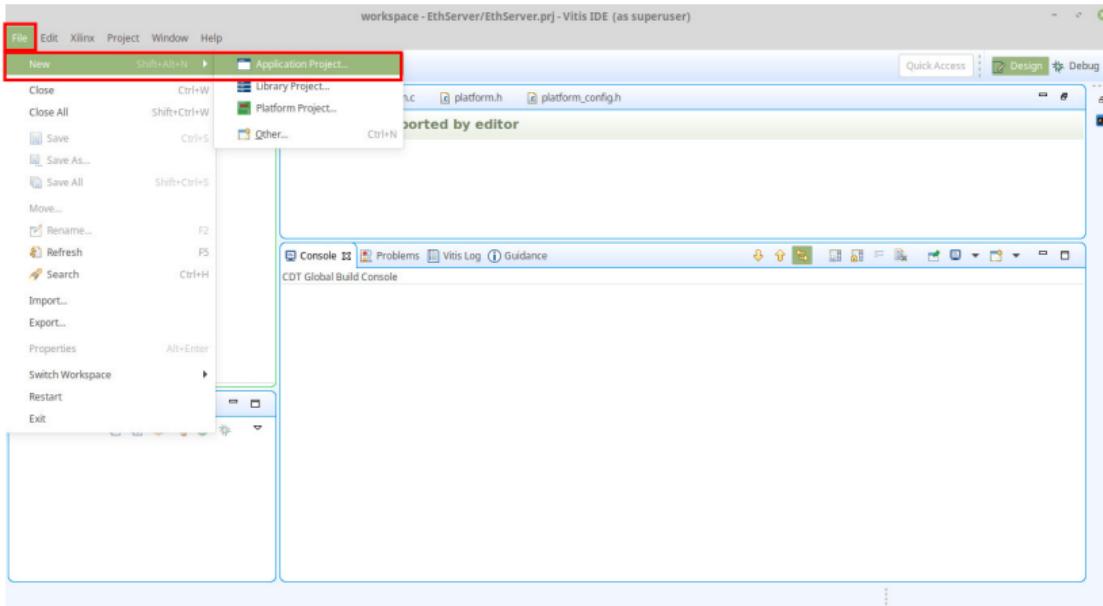
Exportar el hardware desde la opción “File” de la barra de herramientas incluyendo el binario. Por último, ejecutar Vitis.

# Crear la aplicación en Vitis



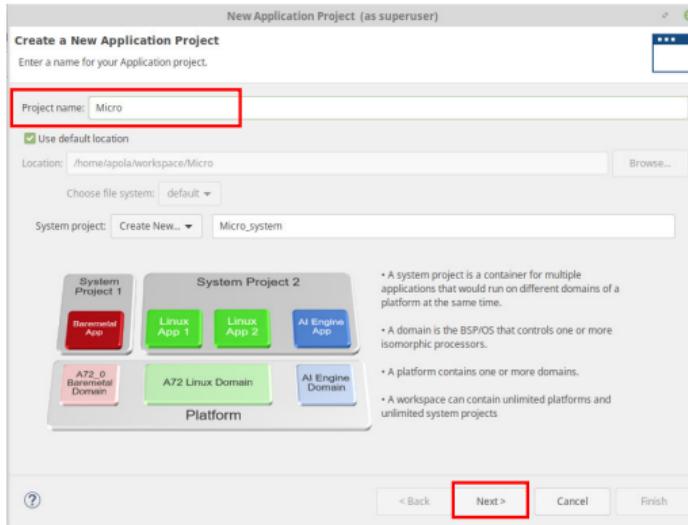
Entorno de trabajo en Vitis. Opción de inicio I: En la página de inicio seleccionar Create Application Project.

# Crear la aplicación en Vitis



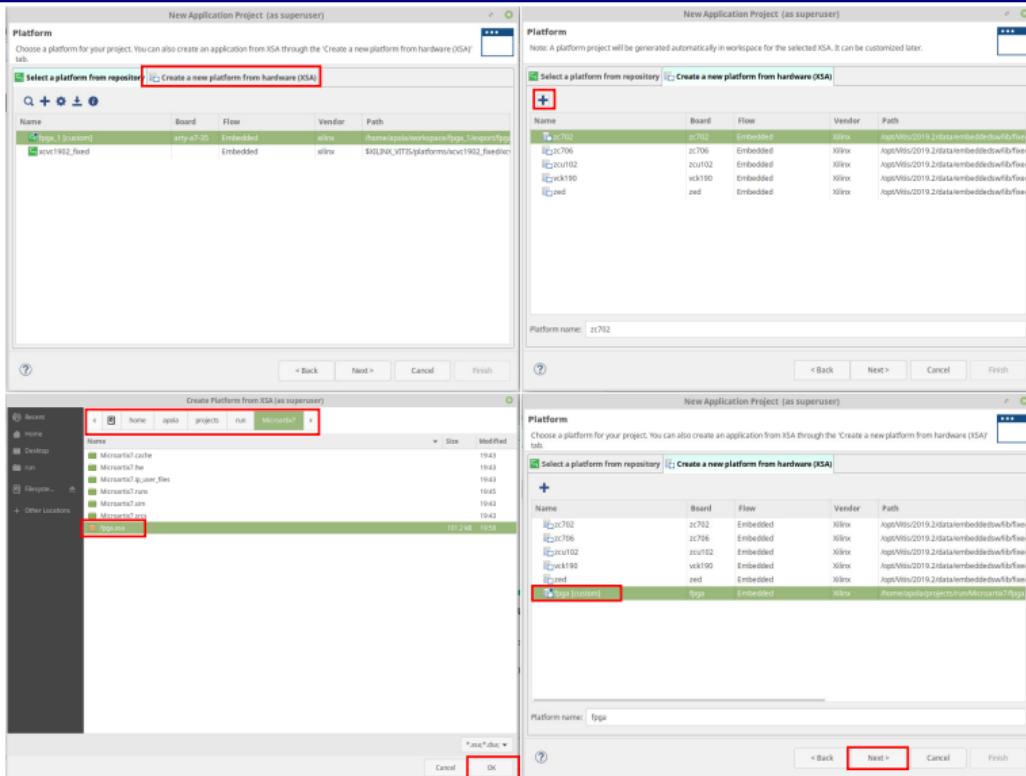
Entorno de trabajo en Vitis. Opción de inicio II: También puede iniciarse la nueva aplicación desde File-New-Application Project.

# Crear la aplicación en Vitis



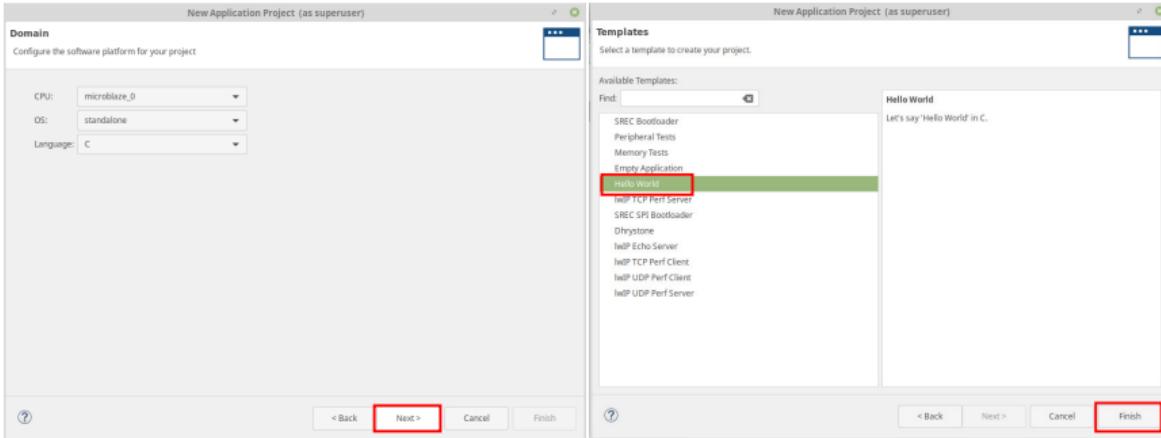
*Asignar nombre al proyecto.*

# Crear la aplicación en Vitis



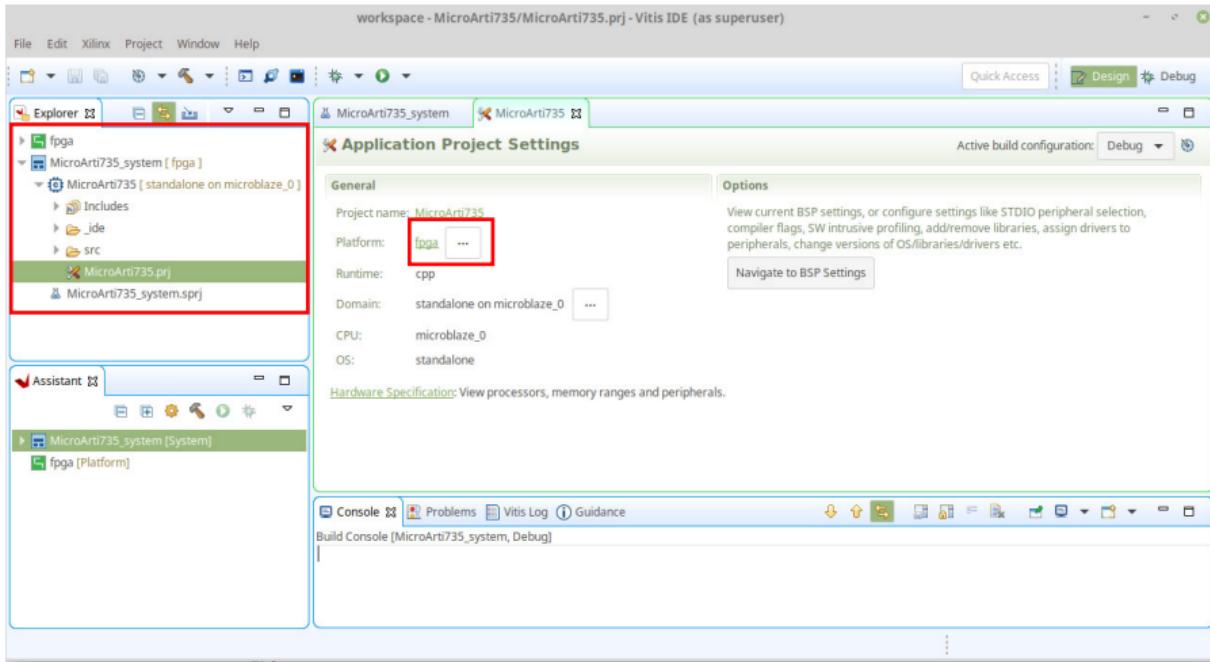
*Crear una nueva plataforma importando el archivo XSA generado con el Vivado.*

# Crear la aplicación en Vitis



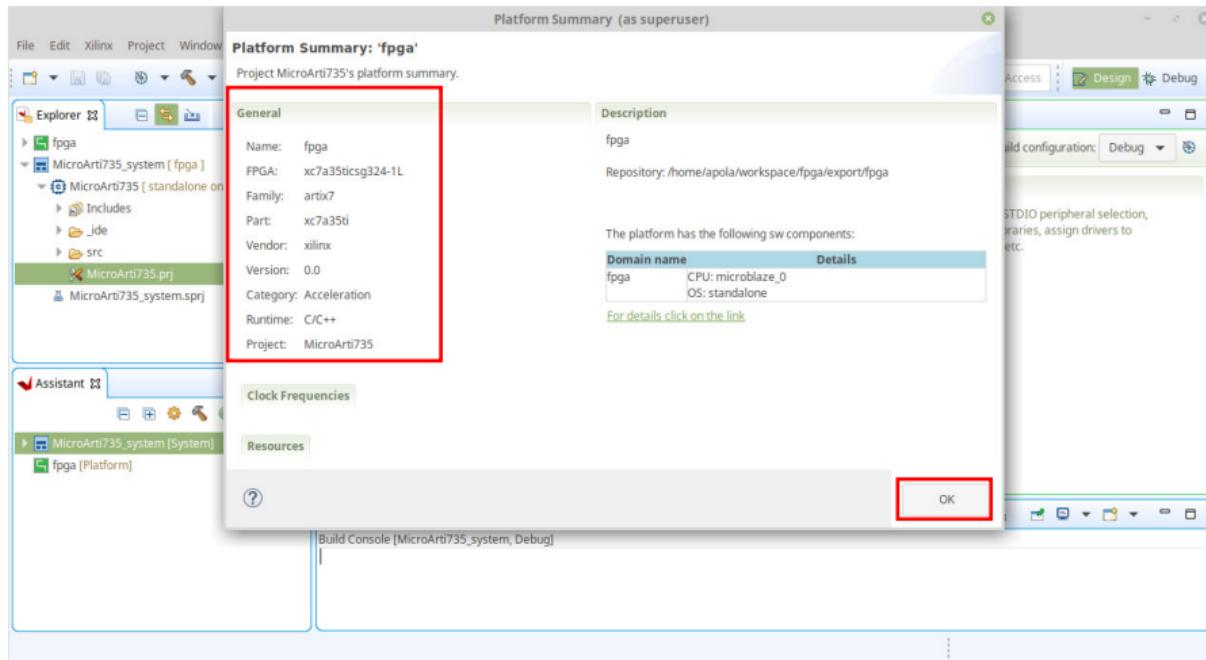
*Verificar las características del micro y asignar el proyecto Hello Word.*

# Crear la aplicación en Vitis



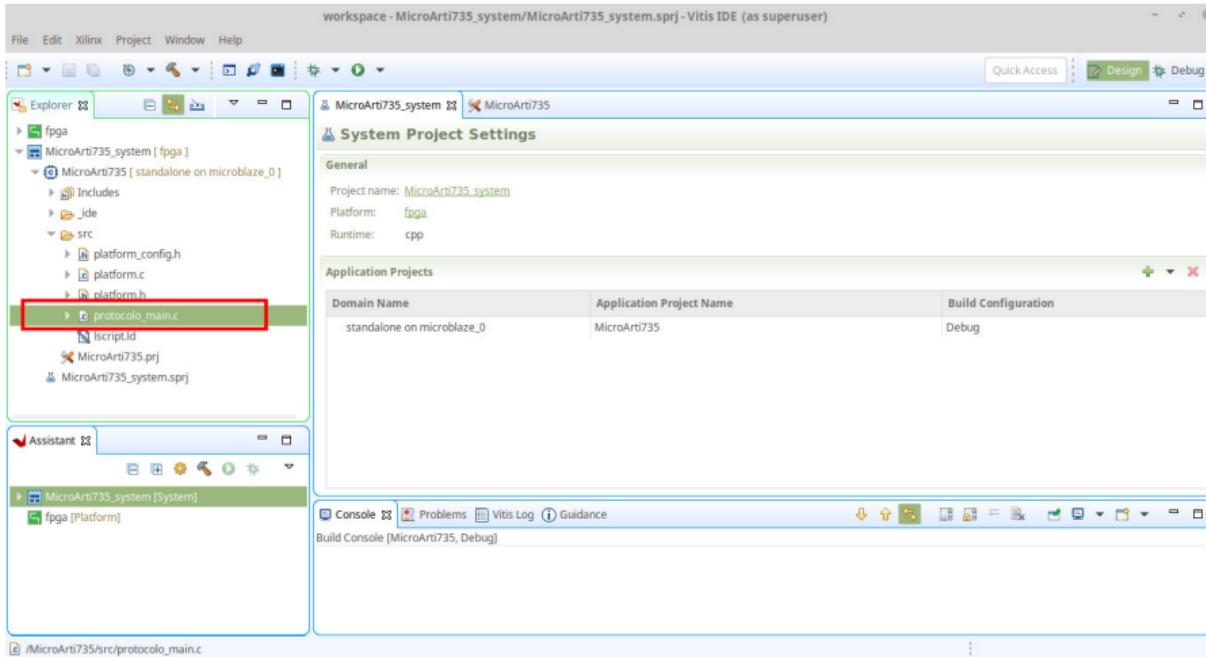
Se generan dos carpetas con el detalle del hardware y la aplicación en C seleccionada. Se puede verificar las características técnicas del diseño haciendo click en el nombre del hardware.

# Crear la aplicación en Vitis



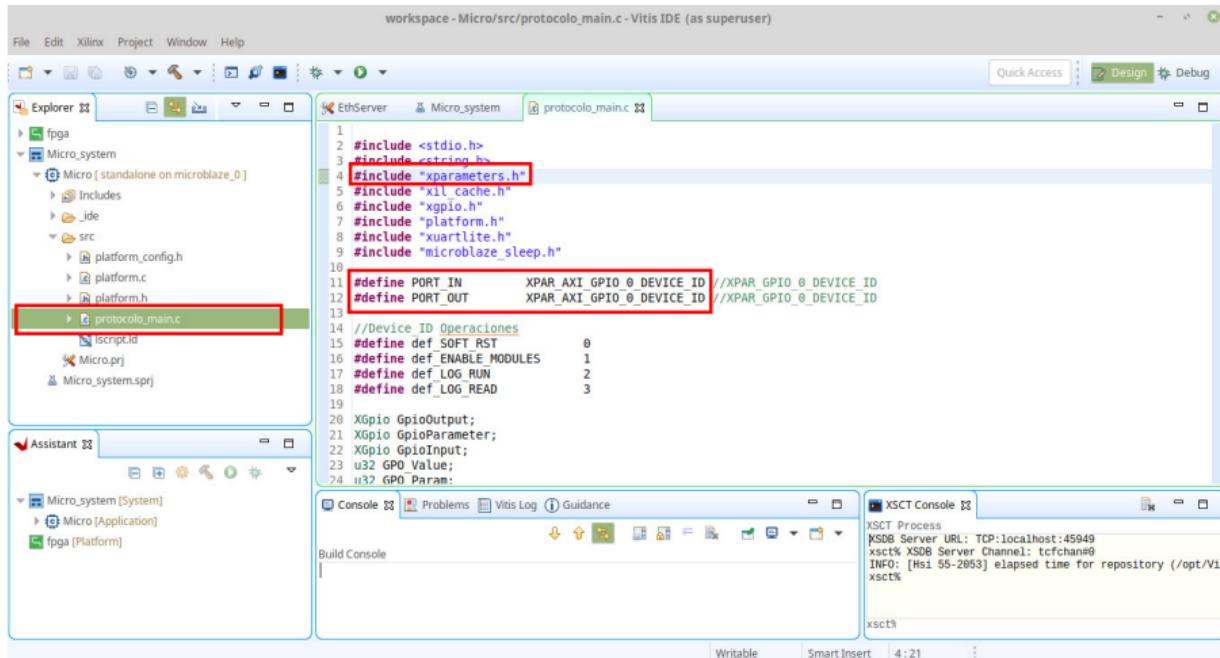
*Detalles de la aplicación.*

# Crear la aplicación en Vitis



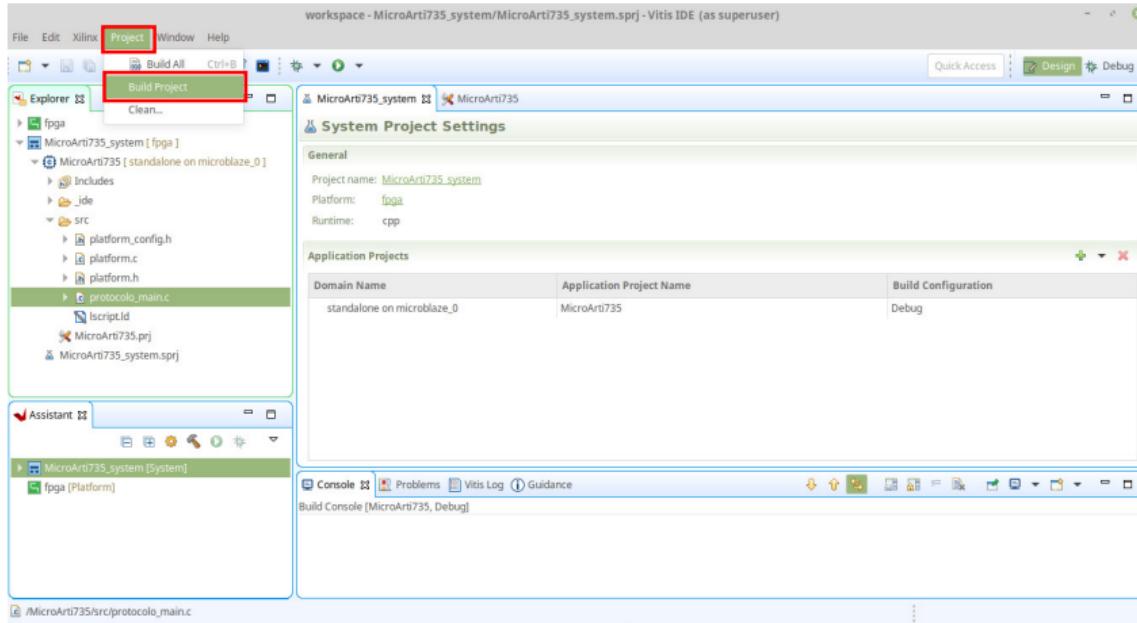
Reemplazar el archivo "Hello Word" con "protocolo\_main" desde el dirección del proyecto. Ej. "/home/apola/workspace/MicroArti735/src/"

# Crear la aplicación en Vitis



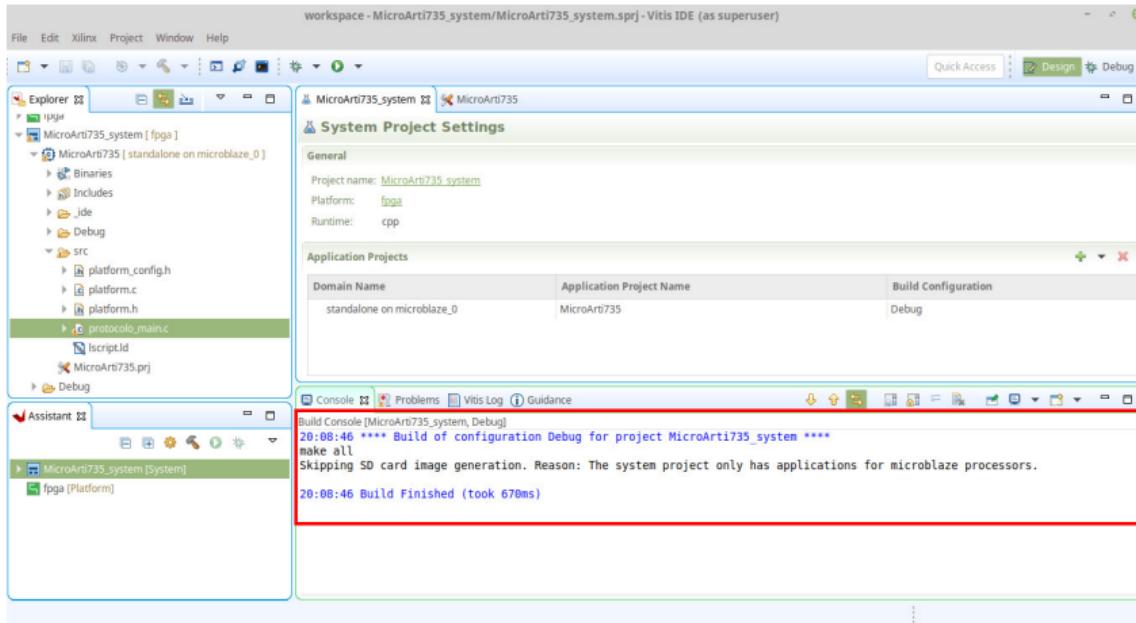
Verificar la definición de los puertos GPIO desde el archivo "xparameters.h". En caso de error en la compilación, debemos buscar en el archivo "xparameters.h" la definición que contenga las palabras claves XPAR, AXI, GPIO, DEVICE e ID.

# Crear la aplicación en Vitis



*Compilar el proyecto (Build Project) o compilar todo Build All).*

# Crear la aplicación en Vitis

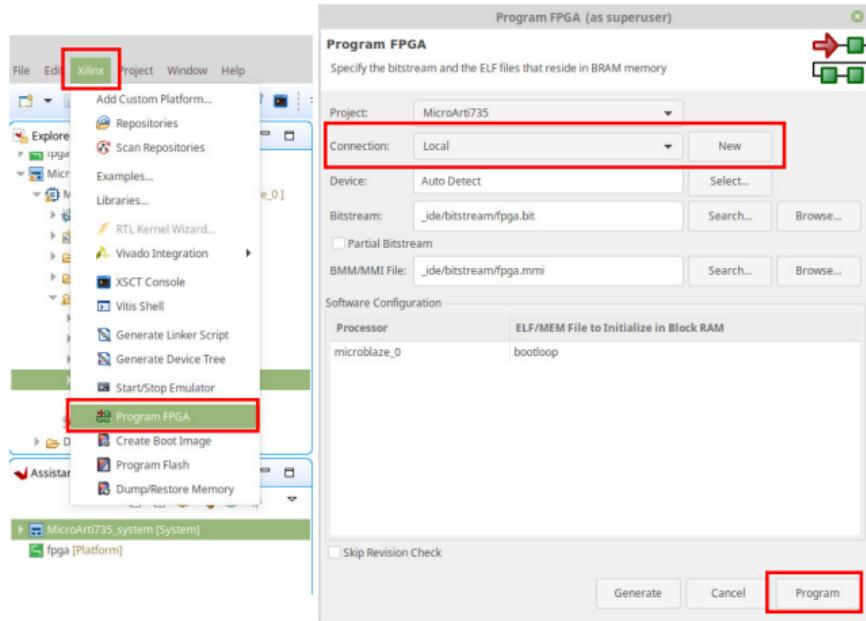


Verificar el estado de la compilación.

## Sección 9

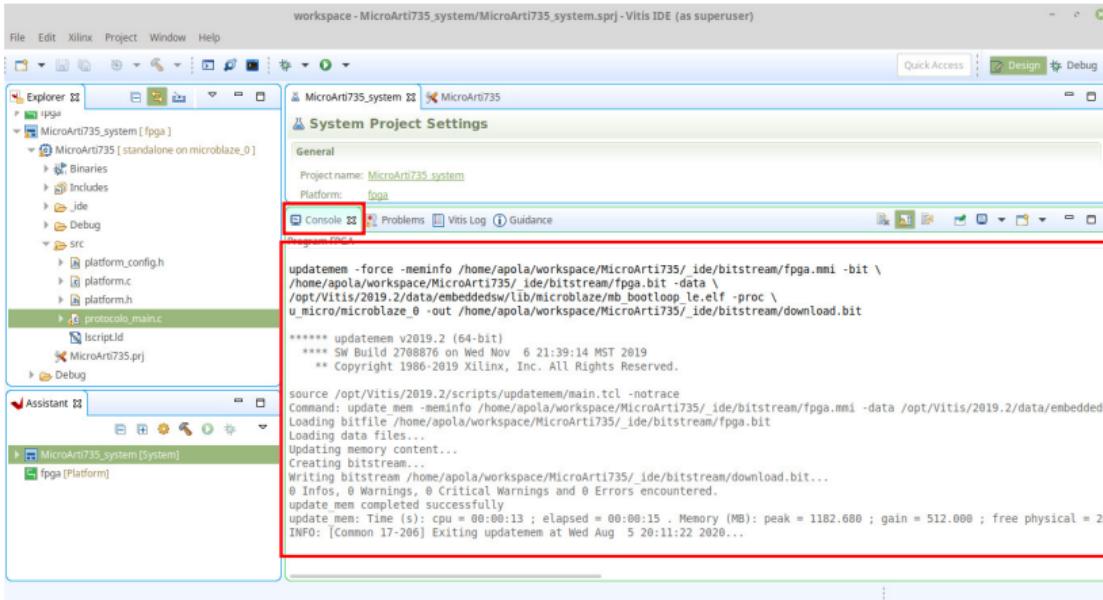
### Programar la FPGA y Ejecutar la Aplicación en Forma Local

# Programar la FPGA



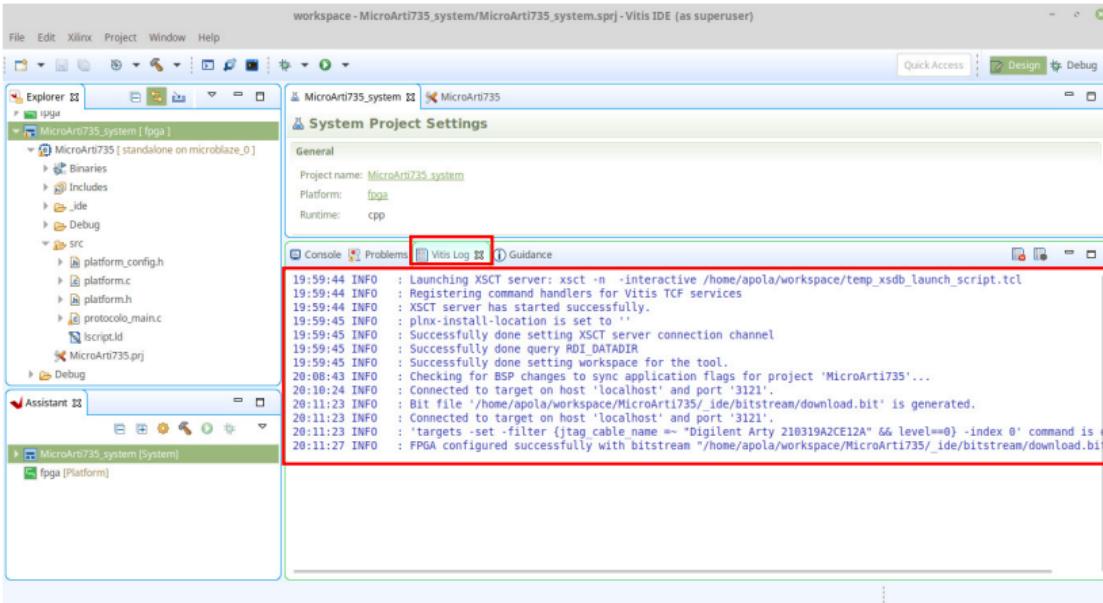
*Programar la FPGA. Chequear que la conexión sea LOCAL.*

# Programar la FPGA



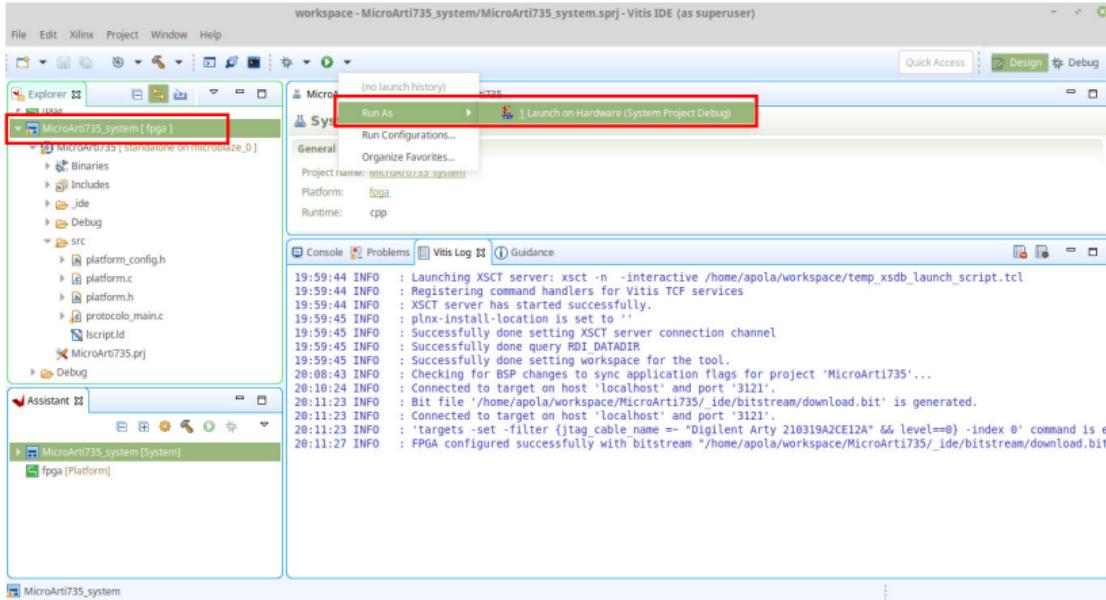
*En la “consola” se observará la generación de un nuevo binario.*

# Programar la FPGA



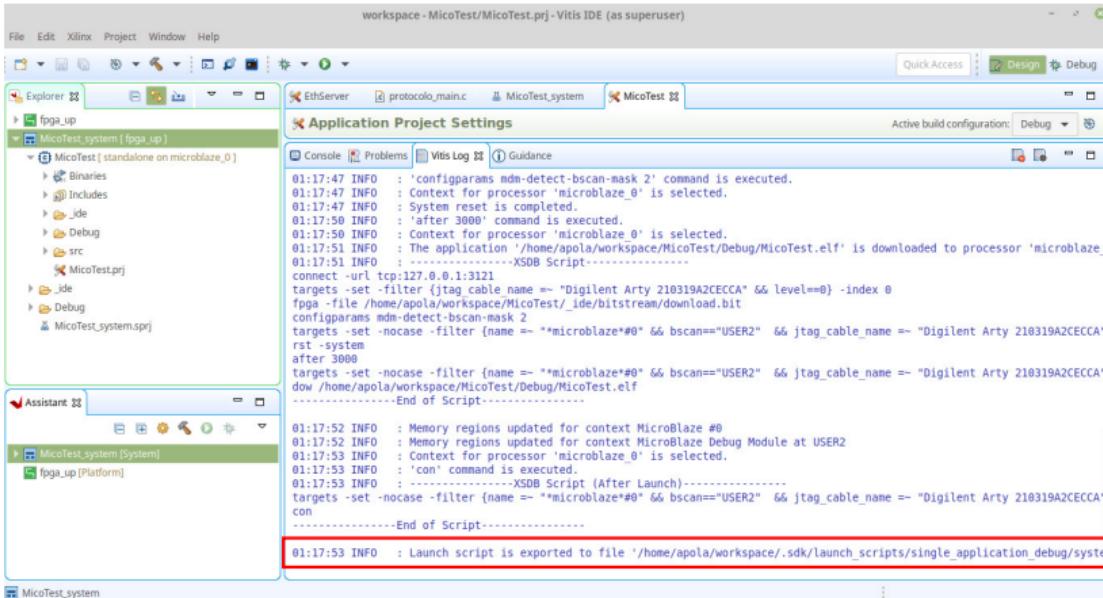
En “Vitis Log” se detallará la programación.

# Ejecutar la Aplicación



*Correr la aplicación en modo Debug. Expandir la opción Run y seleccionar Launch on Hardware. En caso que no salga habilitada la opción se debe volver a hacer click sobre la carpeta del proyecto.*

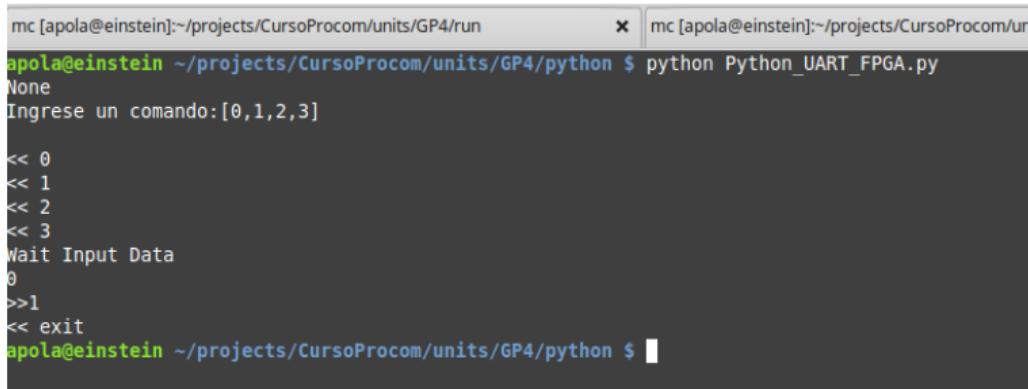
# Ejecutar la Aplicación



*El mensaje Launch verifica que se esta corriendo la aplicación en el microProcesador.*

# Conectar con Aplicación

- Antes de ejecutar, debemos verificar en que puerto se encuentra conectada la FPGA.
- El comando **ls dev** nos permite obtener el puerto al cual esta conectada la FPGA (Ej. ttyUSB1 para Linux - COM3 para Windows).
- Dentro del código del script cambiamos el puerto.
- Ejecutar el archivo Python desde un terminal y verificar que los leds se enciendan.



```
mc [apola@einstein]:~/projects/CursoProcom/units/GP4/run x mc [apola@einstein]:~/projects/CursoProcom/un
apola@einstein ~/projects/CursoProcom/units/GP4/python $ python Python_UART_FPGA.py
None
Ingrese un comando:[0,1,2,3]

<< 0
<< 1
<< 2
<< 3
Wait Input Data
0
>>1
<< exit
apola@einstein ~/projects/CursoProcom/units/GP4/python $
```

Los valores 0, 1 y 2 prenden los leds de diferentes colores y 3 lee el estado de los switch (puede que se apaguen los leds).

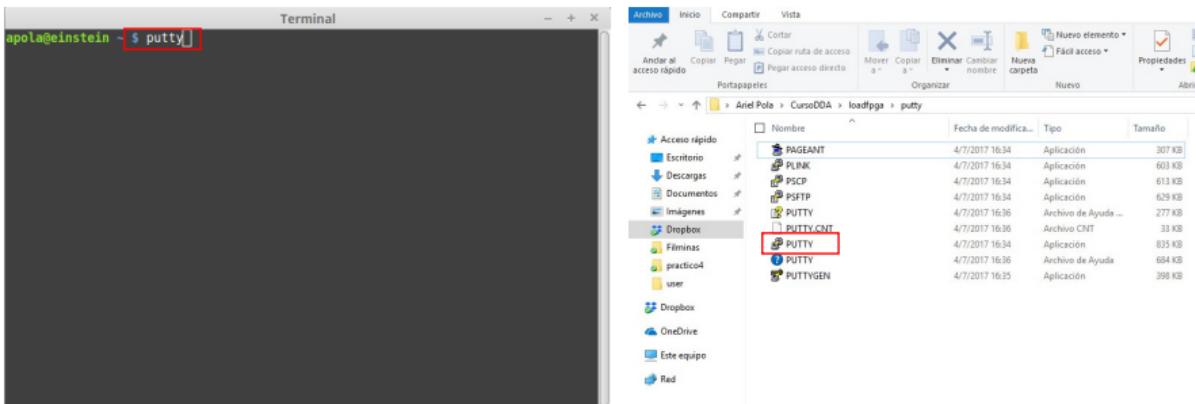
## Sección 10

**Túnel, Programar la FPGA y Ejecutar la Aplicación  
en Forma Remota**

# Túnel y Asignación de FPGA

## Acceso Remoto a Servidor

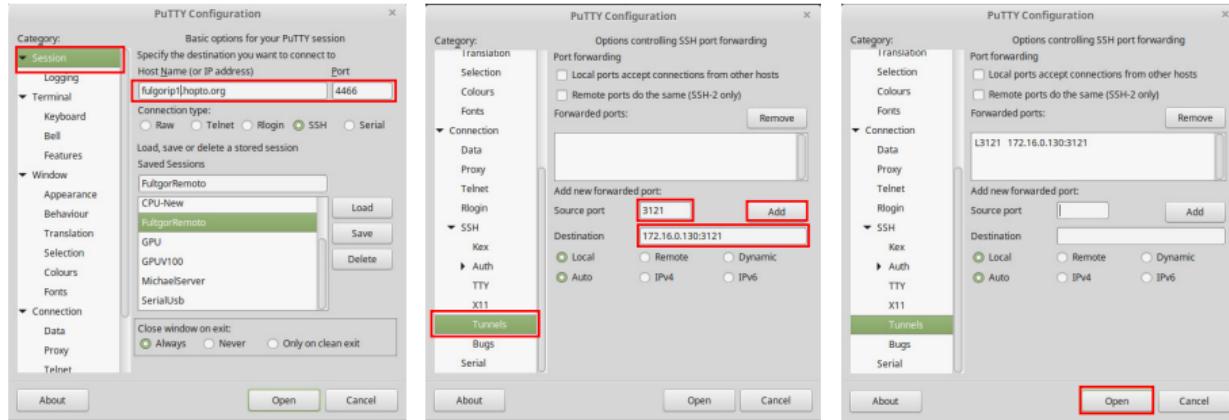
- Instalar la herramienta Putty. En el caso de Windows se tendrán los ejecutables descargados desde <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>.
- Ejecutar
  - **Linux:** En un terminal ejecutar el programa Putty.
  - **Windows:** Hacer doble click sobre el icono de Putty.



Linux

Windows

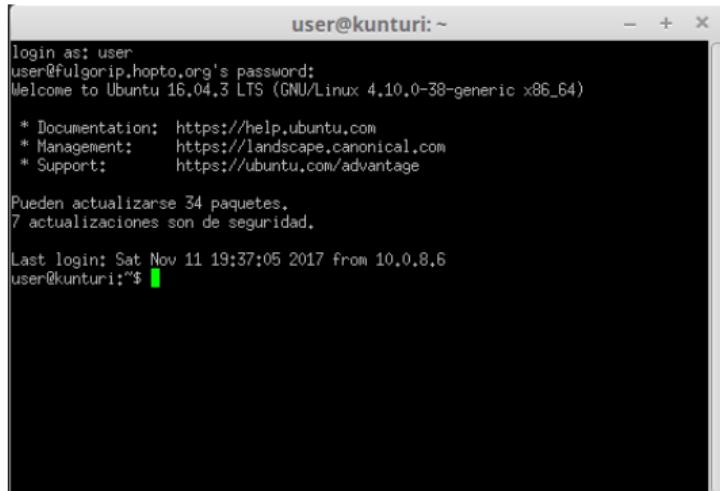
# Túnel y Asignación de FPGA



- En la categoría **Session**, setear el host: *fulgorip.hopto.org* o *fulgorip1.hopto.org* y el puerto: *4466*.
- En la categoría **Connection** –> **SSH** –> **Tunnels** incluir el *forwarded* del puerto y agregarlo a la lista.
- Abrir el túnel.

# Túnel y Asignación de FPGA

- Para el acceso se solicita usuario (**user**) y contraseña (**Cai1keaw**).
- Este acceso es para todos por igual.



```
user@kunturi:~  
login as: user  
user@fulgorip.hopto.org's password:  
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-38-generic x86_64)  
  
 * Documentation: https://help.ubuntu.com  
 * Management: https://landscape.canonical.com  
 * Support: https://ubuntu.com/advantage  
  
Pueden actualizarse 34 paquetes.  
7 actualizaciones son de seguridad.  
  
Last login: Sat Nov 11 19:37:05 2017 from 10.0.8.6  
user@kunturi:~$
```

# Túnel y Asignación de FPGA

- Para generar el entorno de trabajo debemos ejecutar la siguiente linea de comando dentro de la carpeta **work\_dda** agregando el usuario personal con la inicial del nombre y el apellido completo (Ej. apola).

```
1 cd work_dda  
2 python ..//Escritorio/scripts/make_user.py
```

- El script copia unos archivos y arma el árbol de carpetas.

```
user@kunturi:~/work_dda  
login as: user  
user@fulgorip.hopto.org's password:  
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-38-generic x86_64)  
  
 * Documentation: https://help.ubuntu.com  
 * Management: https://landscape.canonical.com  
 * Support: https://ubuntu.com/advantage  
  
Pueden actualizarse 34 paquetes.  
7 actualizaciones son de seguridad.  
  
Last login: Tue Nov 14 13:53:16 2017 from 181.95.86.185  
user@kunturi:~$ cd work_dda/  
user@kunturi:~/work_dda$ python ..//Escritorio/scripts/make_user.py  
Write user (Ex. apola): apola  
Copy: client.py  
Copy: protocolo_uart_dda.py  
Copy: Dec2bin.py  
Copy: DSPTools.py  
End Process  
user@kunturi:~/work_dda$
```

# Túnel y Asignación de FPGA

- Acceder a la carpeta <user>/scripts/ y ejecutar **client.py** para solicitar un puerto USB y el ID de la placa FPGA disponible.

1 *cd apola/scripts*

2 *python client.py*

- El script retorna el USB: **USB3** y el ID de la FPGA: **210319A2CECCA**.
- En caso de no haber disponible ninguna placa, el script retorna que los puertos están ocupados.
- Esta ventana y el script no se tienen que cerrar hasta no terminar de usar la FPGA, ya que se libera el puerto y dará acceso a otro usuario.
- En las siguientes figuras se observa el caso de asignación correcta y puertos ocupados.

## Nota: Copia de archivos o carpetas

- Usamos el comando **pscp** para copiar archivos entre sesiones.

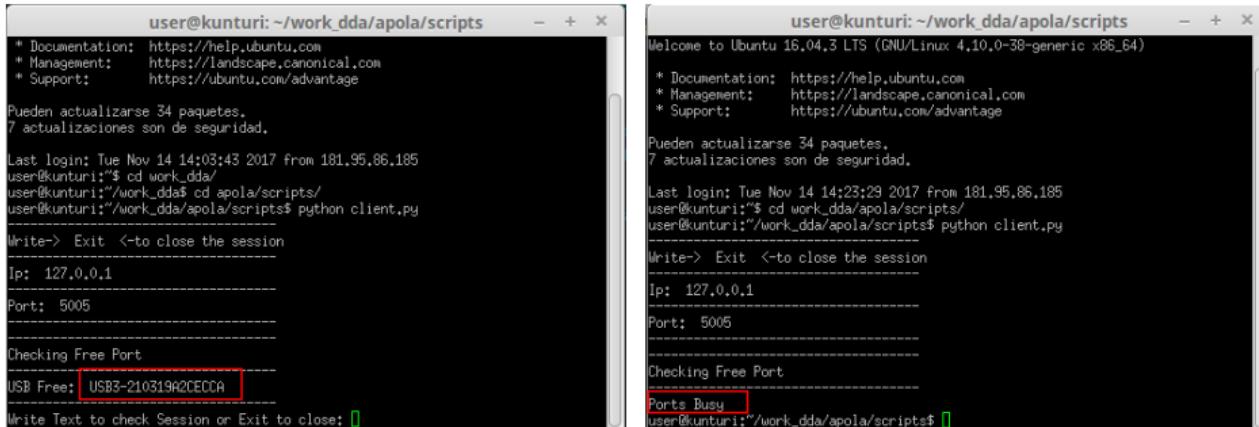
- Remoto a local

```
pscp -P 4466 user@fulgorip1.hopto.org:/home/user/work_dda/<user>/<file> ./
```

- Local a remoto

```
pscp -P 4466 ./<file> user@fulgorip1.hopto.org:/home/user/work_dda/<user>/
```

# Túnel y Asignación de FPGA



The image displays two side-by-side terminal windows from a Linux system (Ubuntu 16.04.3 LTS). Both terminals show the same command-line session, which includes documentation links, a package update notice, a login history, and a port assignment process.

**Terminal Left (Port Assignment):**

```
user@kunturi:~/work_dda/apola/scripts
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

Pueden actualizarse 34 paquetes,
7 actualizaciones son de seguridad.

Last login: Tue Nov 14 14:03:43 2017 from 181.95.86.185
user@kunturi:$ cd work_dda/
user@kunturi:~/work_dda$ cd apola/scripts/
user@kunturi:~/work_dda/apola/scripts$ python client.py

Write-> Exit <-to close the session

Ip: 127.0.0.1
-----
Port: 5005
-----
Checking Free Port
USB Free: USB3-210319A20ECC0
-----
Write Text to check Session or Exit to close: 
```

**Terminal Right (Port Usage):**

```
user@kunturi:~/work_dda/apola/scripts
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-38-generic x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

Pueden actualizarse 34 paquetes,
7 actualizaciones son de seguridad.

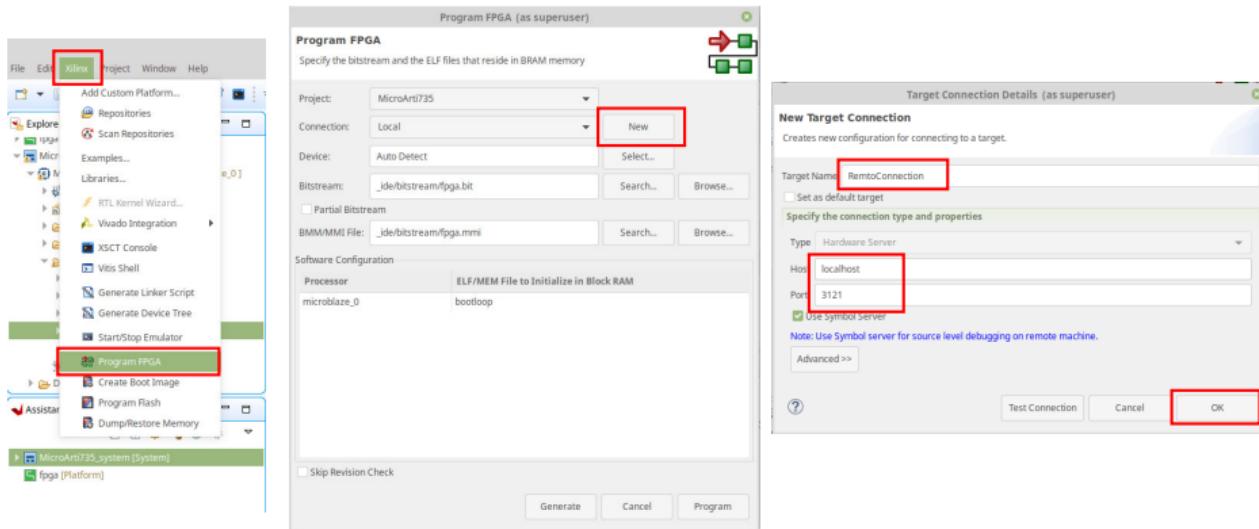
Last login: Tue Nov 14 14:23:29 2017 from 181.95.86.185
user@kunturi:$ cd work_dda/apola/scripts/
user@kunturi:~/work_dda/apola/scripts$ python client.py

Write-> Exit <-to close the session

Ip: 127.0.0.1
-----
Port: 5005
-----
Checking Free Port
Ports Busy
user@kunturi:~/work_dda/apola/scripts$ 
```

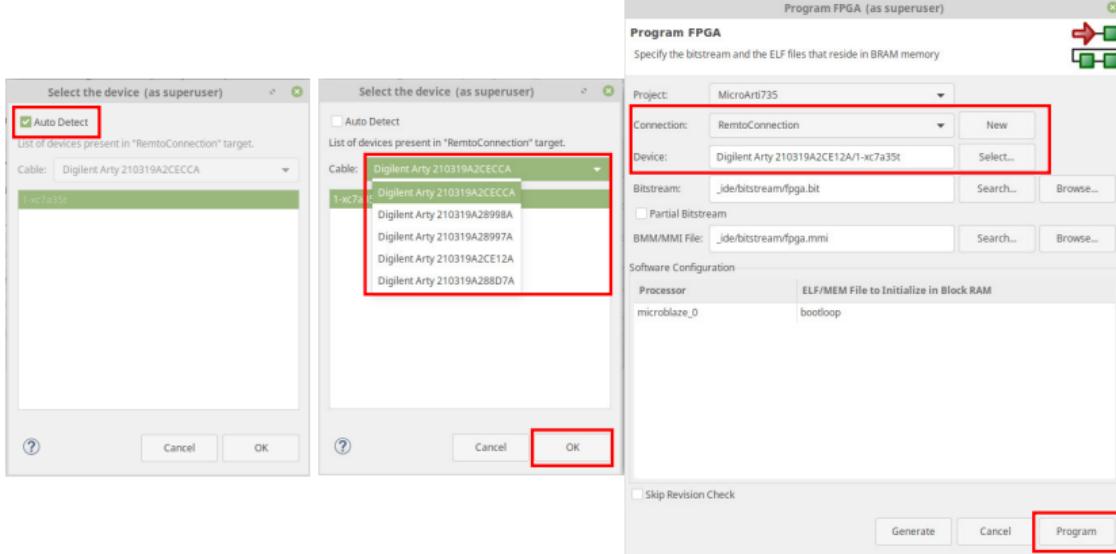
Las figuras muestran la asignación correcta de puerto (izq.) y los puertos ocupados (der.).

# Programar la FPGA



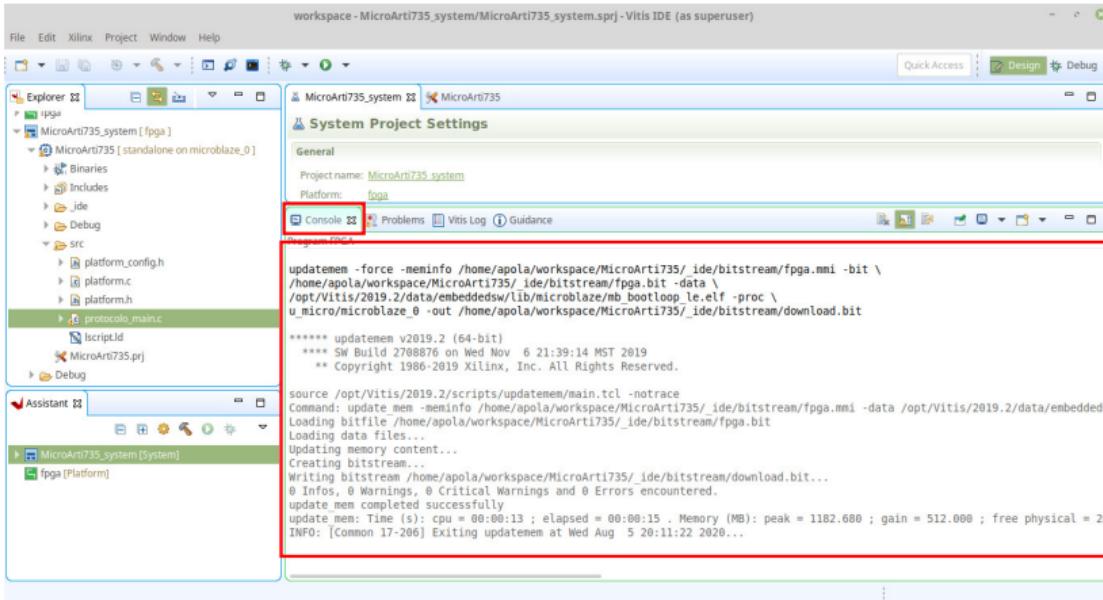
*Programar la FPGA. Crear un nuevo Server, dar un nombre y colocar la dirección IP del server remoto. Para nuestro caso, el cual realiza un forward del puerto 3121 se debe colocar LOCALHOST como dirección de host.*

# Programar la FPGA



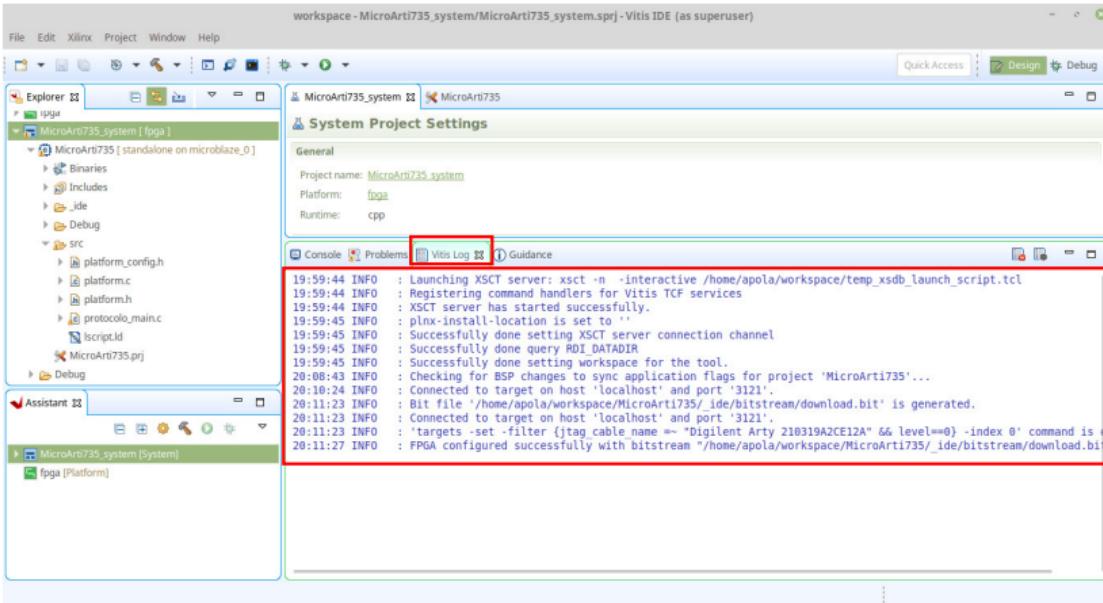
Desactivar AutoDetect, seleccionar el ID de la FPGA y verificar la información.

# Programar la FPGA



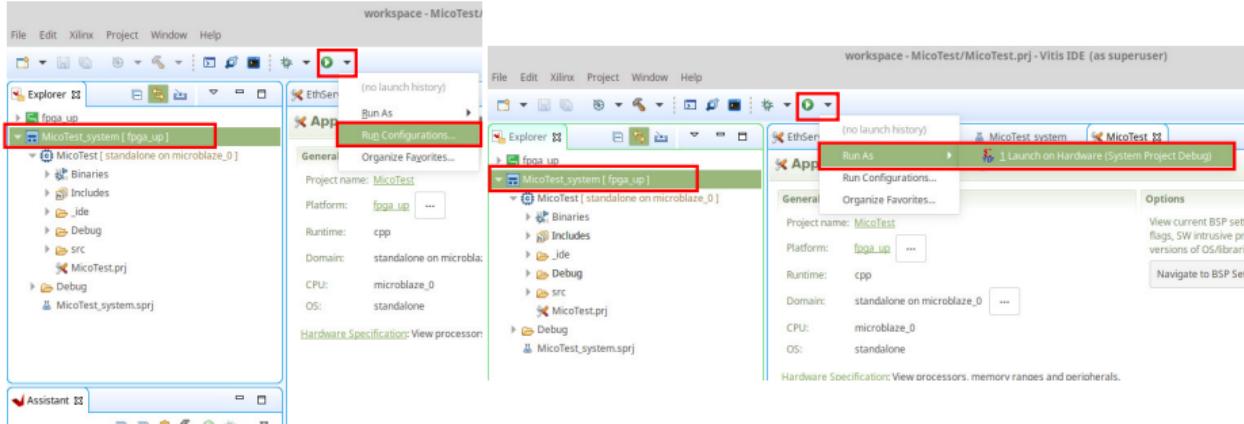
*En la “consola” se observará la generación de un nuevo binario.*

# Programar la FPGA



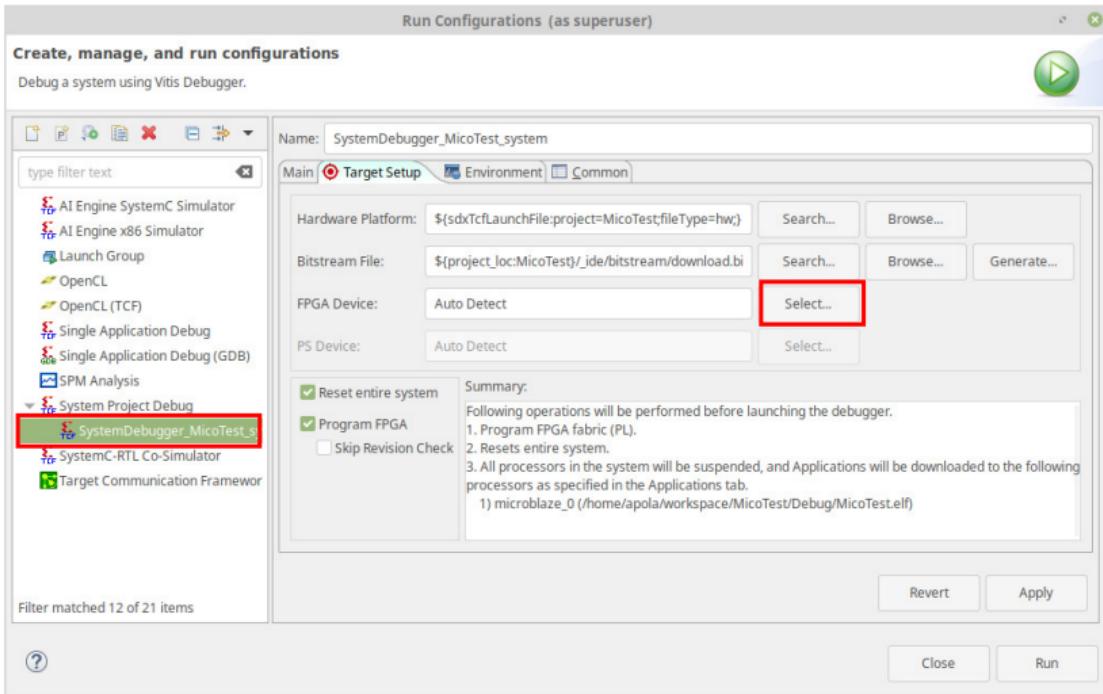
En “Vitis Log” se detallará la programación.

# Ejecutar la Aplicación



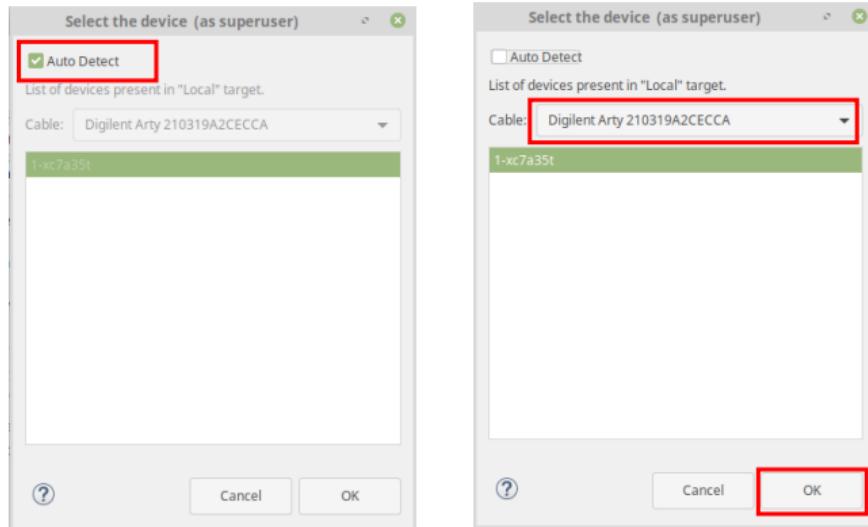
Antes de ejecutar la aplicación es necesario setear el dispositivo que se quiere correr. Para ello necesitamos acceder a *Run Configuration*. Nota: En caso que no aparezcan las opciones que se detallan en la figura de la siguiente pagina, debemos ejecutar al menos una vez la aplicación. En este caso debemos seguir los pasos de la figura de la derecha.

# Ejecutar la Aplicación



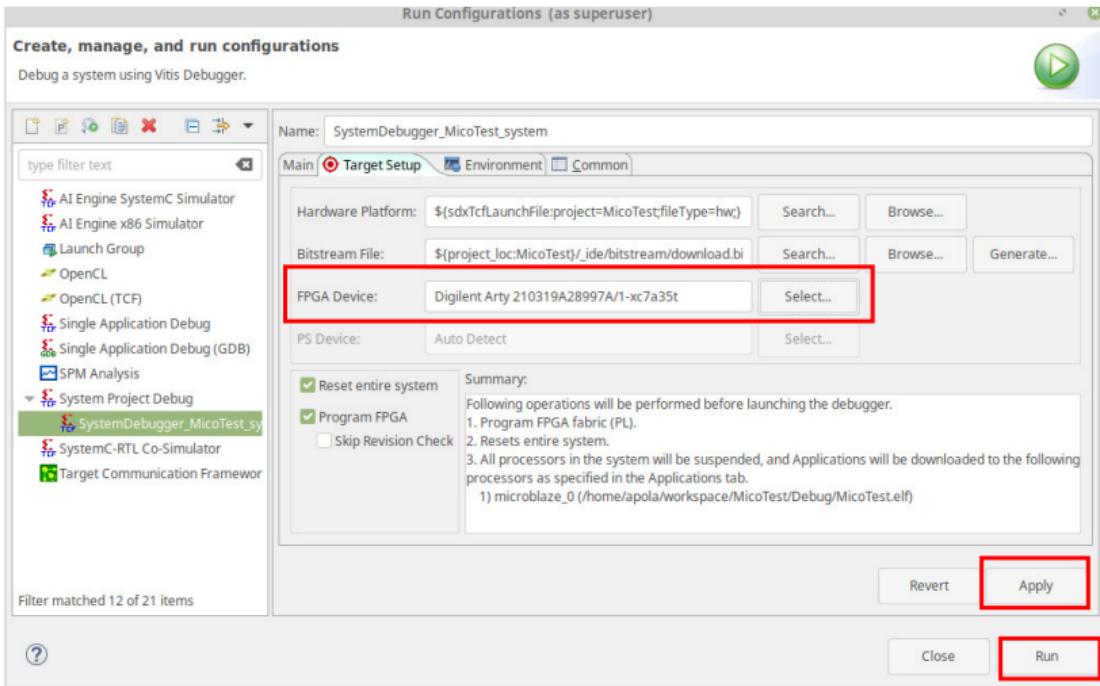
*En la opción SystemDebugger, cambiar el dispositivo (FPGA Device).*

# Ejecutar la Aplicación



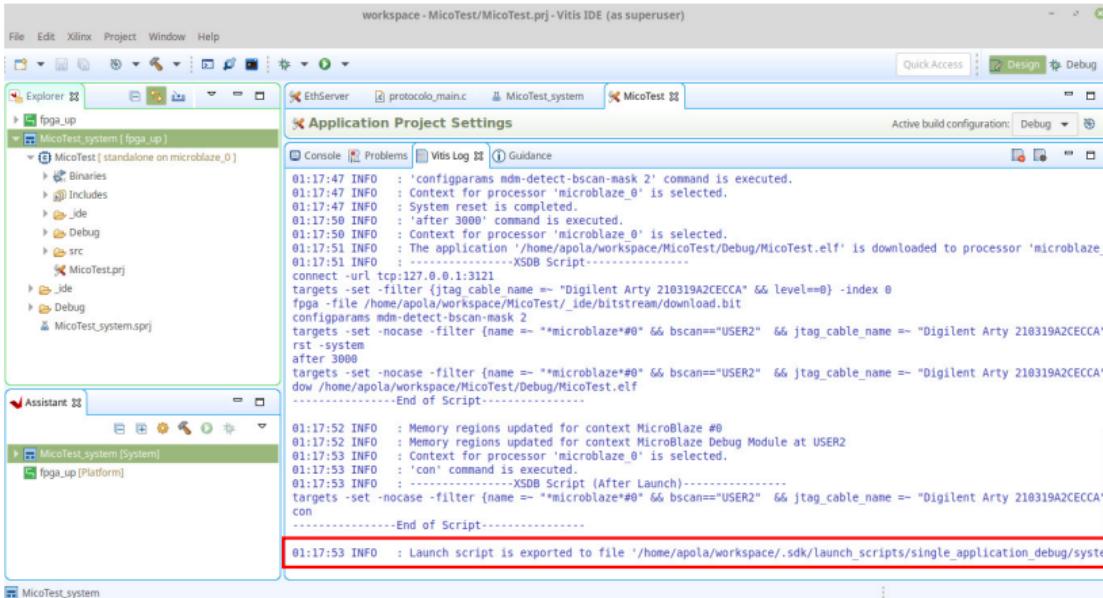
*Seleccionar una FPGA.*

# Ejecutar la Aplicación



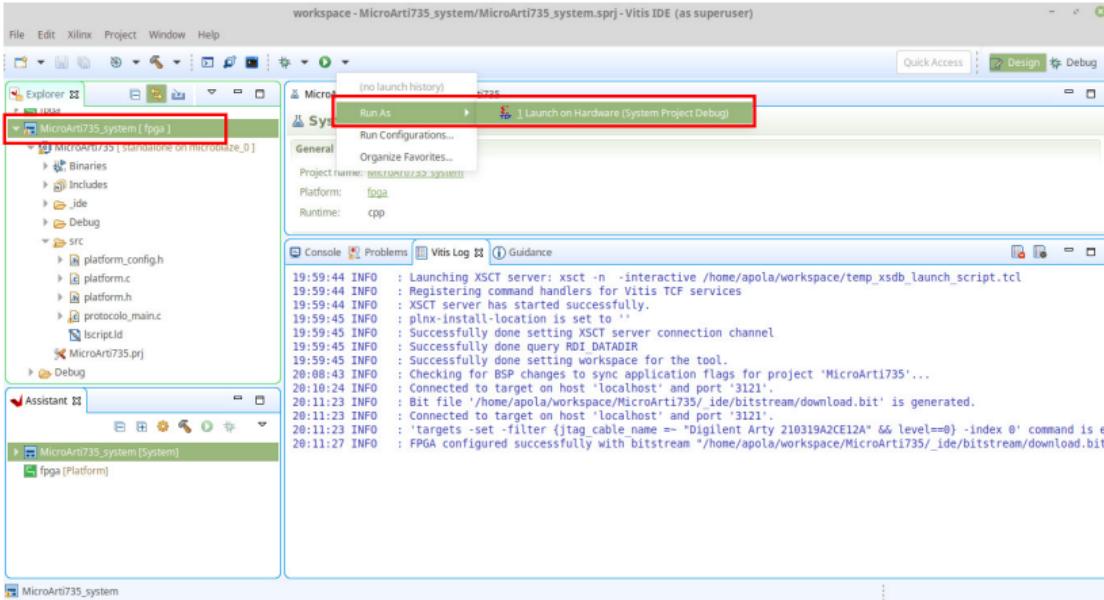
*Verificar el ID, aplicar los cambios y ejecutar.*

# Ejecutar la Aplicación



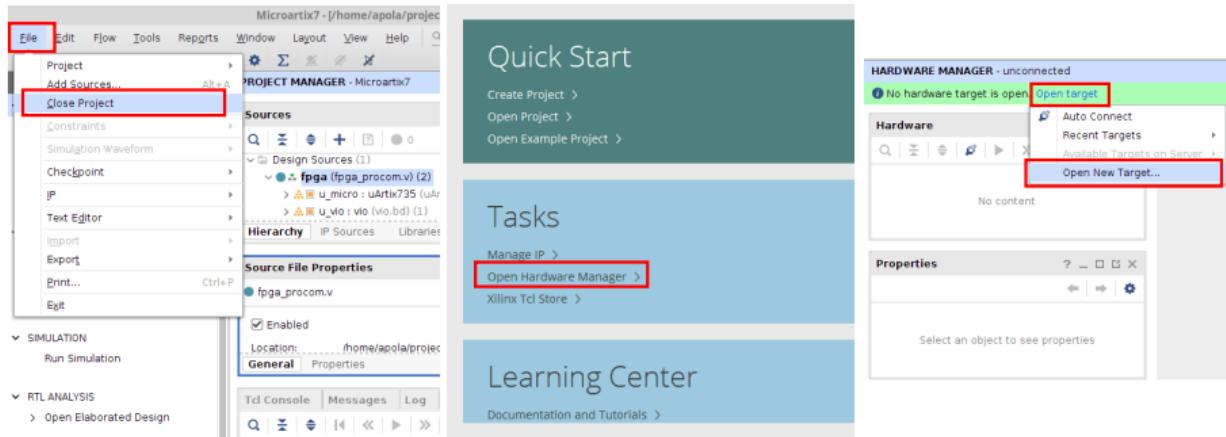
*El mensaje Launch verifica que se esta corriendo la aplicación en el microprocesador.*

# Ejecutar la Aplicación



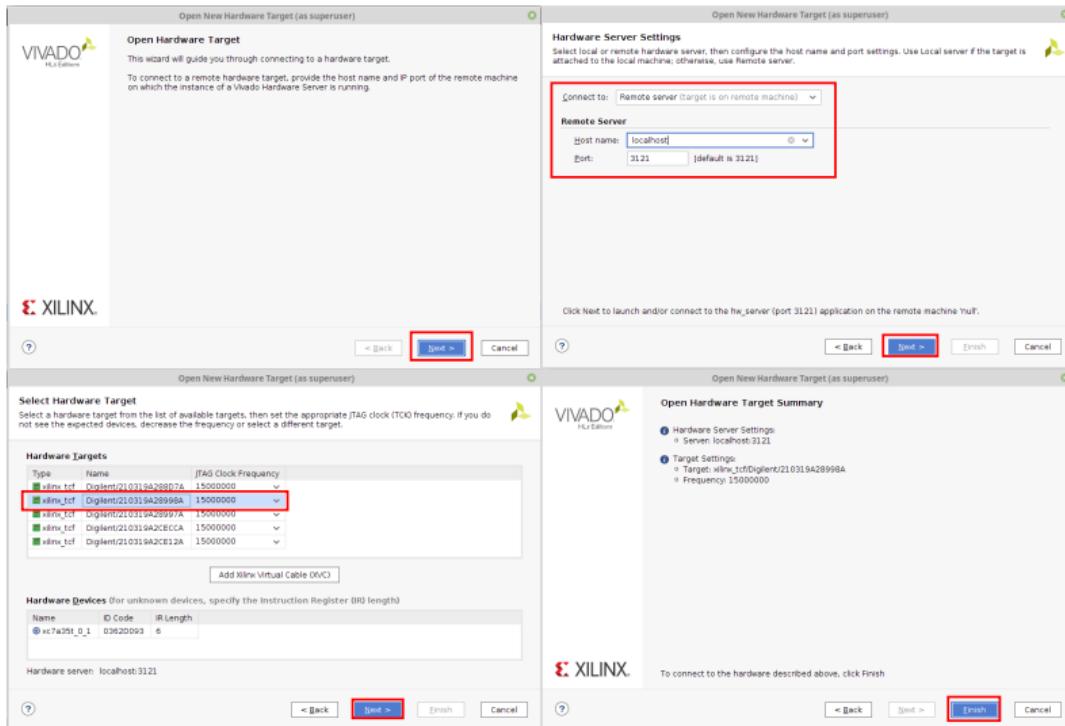
*En caso de necesitar ejecutar varias veces la aplicación y una vez que se han ejecutado los pasos anteriores, basta con lanzar nuevamente la aplicación.*

# Cargar parámetros del VIO



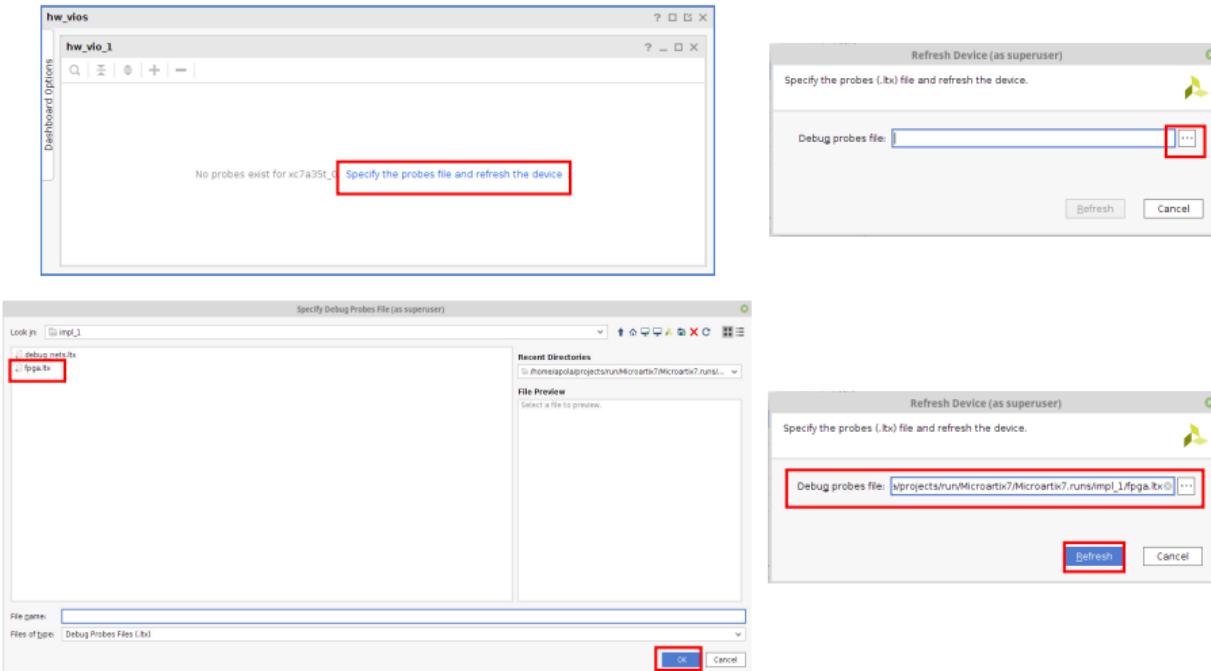
*Volver a la ventana del Vivado, cerrar el proyecto, abrir el controlador de hardware y seleccionar un nuevo target.*

# Cargar parámetros del VIO



Crear un nuevo servidor cuya IP sea LOCALHOST y seleccionar la FPGA con el ID que previamente el servidor les entregó.

# Cargar parámetros del VIO



Debemos agregar el archivo con extensión ltx que se generó cuando creamos el VIO en los pasos anteriores. Se encuentra dentro de la carpeta <nameProject>.runs/impl\_1/<nameBinario>.ltx

# Cargar parámetros del VIO

The screenshot shows the Vivado interface for managing VIO probes. On the left, the 'hw\_vios' dashboard for 'hw\_vio\_1' is displayed, with a red box highlighting the '+' button to add probes. In the center, the 'Add Probes' dialog is open, listing several probe options. A red box highlights the list of probes: u\_violprobe\_in0\_0[2:0], u\_violprobe\_in1\_0[2:0], u\_violprobe\_in2\_0[2:0], u\_violprobe\_in3\_0[2:0], u\_violvio\_0\_probe\_out0, and u\_violvio\_0\_probe\_out1[3:0]. On the right, the configuration table for the selected probes is shown, with a red box highlighting the row for 'u\_violvio\_0\_probe\_out0'. The table has columns for Name, Value, Activity, Direction, and VIO. The row for 'u\_violvio\_0\_probe\_out0' has a value of [B] 1, activity of [H] 1, direction of Output, and VIO hw\_vio\_1.

Name	Value	Activity	Direction	VIO
u_violprobe_in3_0[2:0]	[H] 1		Input	hw_vio_1
u_violprobe_in2_0[2:0]	[H] 1		Input	hw_vio_1
u_violprobe_in1_0[2:0]	[H] 1		Input	hw_vio_1
u_violprobe_in0_0[2:0]	[H] 1		Input	hw_vio_1
u_violvio_0_probe_out0	[B] 1		Output	hw_vio_1
u_violvio_0_probe_out1[3:0]	[H] 0		Output	hw_vio_1

Agregamos los puertos de entrada y salida que queremos controlar y observar.

# Asignar el Puerto USB

```
login as: user
user@fulgorip1.hopto.org's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.15.0-112-generic x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/advantage

Pueden actualizarse 225 paquetes,
6 actualizaciones son de seguridad.

New release '18.04.5 LTS' available,
Run 'do-release-upgrade' to upgrade to it.

Last login: Sun Aug 16 20:03:13 2020 from 181.31.160.204
user@kuntrui:$ cd work_dda/apola/scripts/
user@kuntrui:/work_dda/apola/scripts$ emacs Python_UART_FPGA.py
```

```
File Edit Options Buffers Tools Python Help
Import time
Import serial

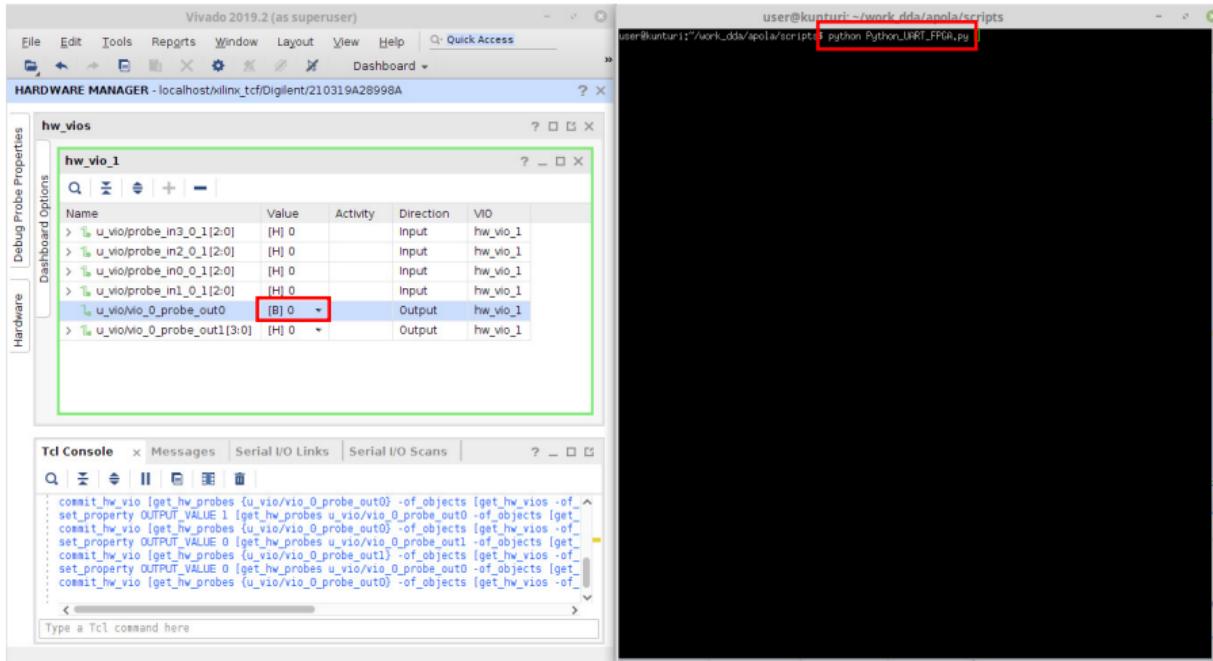
ser = serial.Serial(
    port='/dev/ttyUSB0',          #Configurar con el puerto
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS
)

ser.isOpen()
ser.timeout=None
print(ser.timeout)

print "Ingrese un comando:[0,1,2,3]\r\n"
while 1:
    input = raw_input("=> ")
    if input == 'exit':
        ser.close()
        exit()
    elif input == '3':
        print "Wait Input Data"
        ser.write(input)
        time.sleep(2)
        out = ord(ser.read(1))
        print(ser.inWaiting())
        if out != '':
            print '>>>' + str(out)
    else:
        ser.write(input)
        time.sleep(1)
```

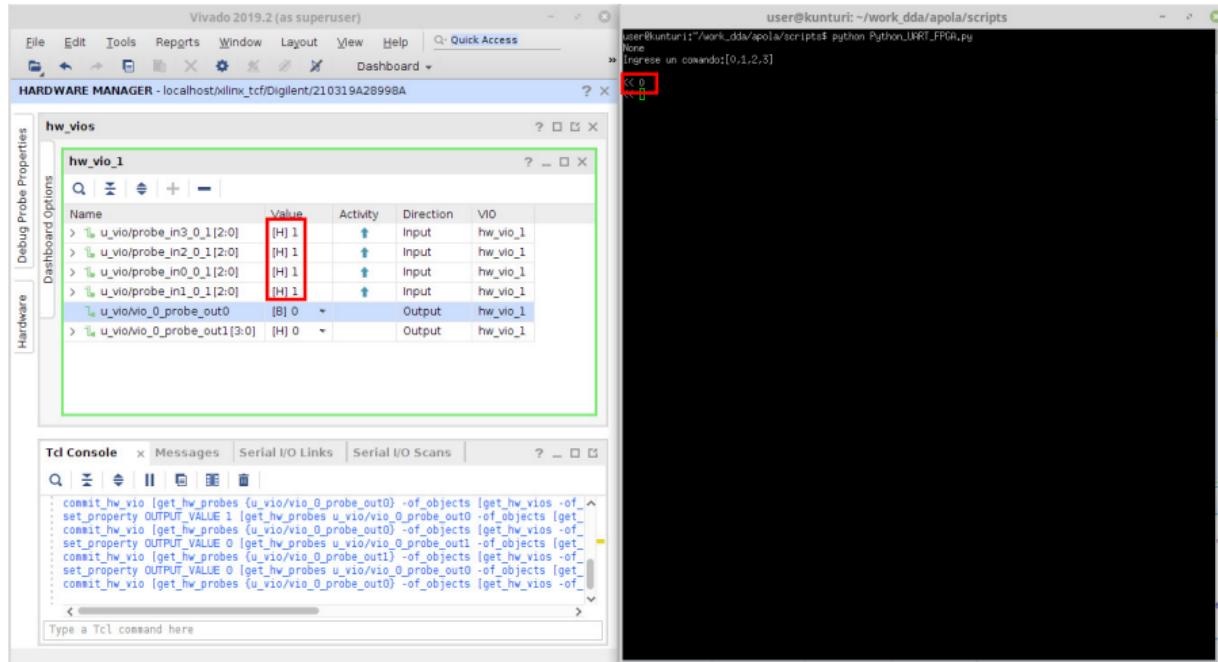
En un nuevo terminal de PUTTY, modificamos el archivo *Python\_UART\_FPGA.py* usando *emacs* u otro editor de texto. Debemos colocar el USB que nos asignaron previamente al momento de conectarnos. Guardar Cambios: *CTRL+x + CTRL+s*. Cerrar *emacs*: *CTRL+x + CTRL+c*

# VIO y Python



El puerto remarcado en el VIO es el que habilita si leemos el estado de los switch y lo que se carga desde el VIO (0: VIOCctrlEnb - 1: Switch). Ejecutamos el python.

# VIO y Python



# VIO y Python

The screenshot shows the Vivado 2019.2 interface with two main windows:

- Hardware Manager - localhost/xilinx\_tcf/Diligent/210319A2B998A**: This window displays the **hw\_vios** table. A specific row for the probe `u_vio/probe_out0` is selected and highlighted with a blue border. The "Value" column for this row is currently set to [H] 2, which is highlighted with a red box. Other rows show values like [H] 2 for `u_vio/probe_in3_0[2:0]`, `u_vio/probe_in2_0[2:0]`, and `u_vio/probe_in0_0[2:0]`, and [B] 0 for `u_vio/vio_0_probe_out0`.
- Tcl Console**: This window shows Python code for interacting with the VIO probes. The code uses the `get_hw_probes` command to get objects for `u_vio/vio_0_probe_out0`, then sets properties for `OUTPUT_VALUE` and `COMMIT`. It also prints the current value of `u_vio/vio_0_probe_out0` and `u_vio/vio_0_probe_out1`.

*Elegimos 0 y activa el color Rojo (1).*

# VIO y Python

The screenshot shows two windows side-by-side. On the left is the Vivado 2019.2 interface, specifically the Hardware Manager. A table titled 'hw\_vios' is displayed, showing various I/O probe settings. One row, 'u\_vio/probe\_in3\_0[2:0]', has its value set to [H] 4, which is highlighted with a red box. On the right is a terminal window with the command 'python Python\_UVIO\_FPGA.py' running. The terminal output shows the command 'Ingresar un comando:[0,1,2,3]' followed by four green arrows pointing down, with the second arrow also highlighted with a red box.

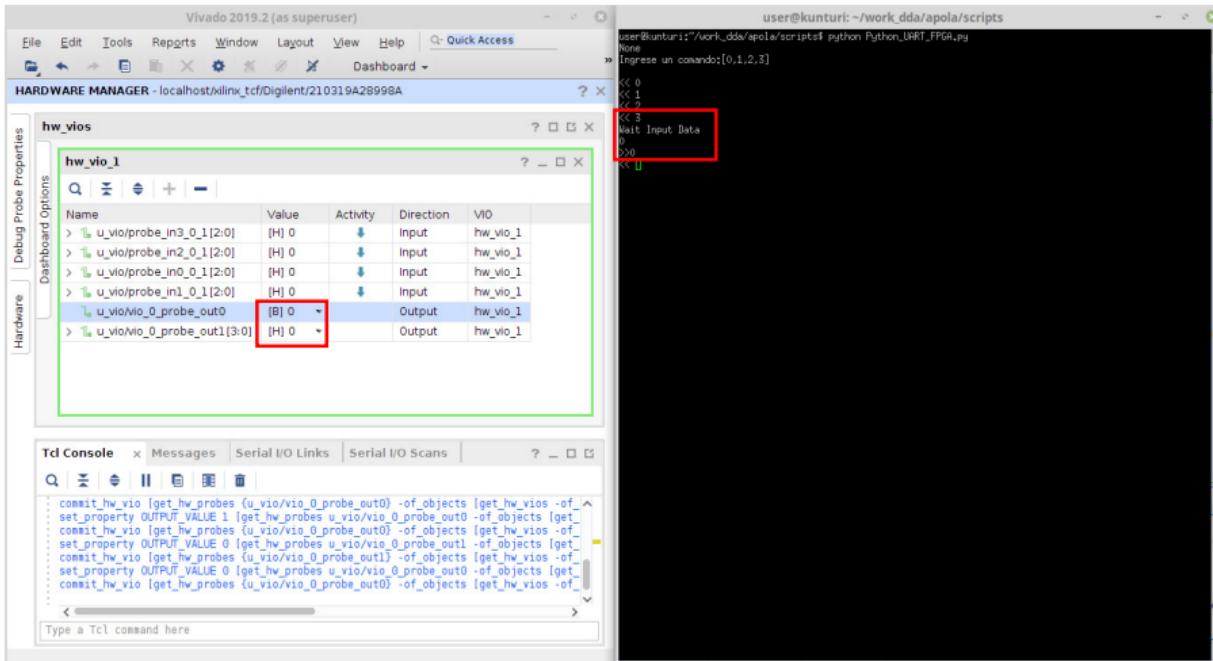
Name	Value	Activity	Direction	VIO
> u_vio/probe_in3_0[2:0]	[H] 4	idle	Input	hw_vio_1
> u_vio/probe_in2_0[2:0]	[H] 4	idle	Input	hw_vio_1
> u_vio/probe_in0_0[2:0]	[H] 4	idle	Input	hw_vio_1
> u_vio/probe_in1_0[2:0]	[H] 4	idle	Input	hw_vio_1
u_vio/vio_0_probe_out0	[B] 0	idle	Output	hw_vio_1
u_vio/vio_0_probe_out1[3:0]	[H] 0	idle	Output	hw_vio_1

Tcl Console

```
commit_hw_vio [get_hw_probes {u_vio/vio_0_probe_out0}] -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out0}];  
set_property OUTPUT_VALUE 1 [get_hw_probes {u_vio/vio_0_probe_out0} -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out0}]]  
commit_hw_vio [get_hw_probes {u_vio/vio_0_probe_out1}] -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out1}];  
set_property OUTPUT_VALUE 1 [get_hw_probes {u_vio/vio_0_probe_out1} -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out1}]]  
commit_hw_vio [get_hw_probes {u_vio/vio_0_probe_out1}] -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out1}];  
set_property OUTPUT_VALUE 0 [get_hw_probes {u_vio/vio_0_probe_out0} -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out0}]]  
commit_hw_vio [get_hw_probes {u_vio/vio_0_probe_out0}] -of_objects [get_hw_vios -of_objects {u_vio/vio_0_probe_out0}];
```

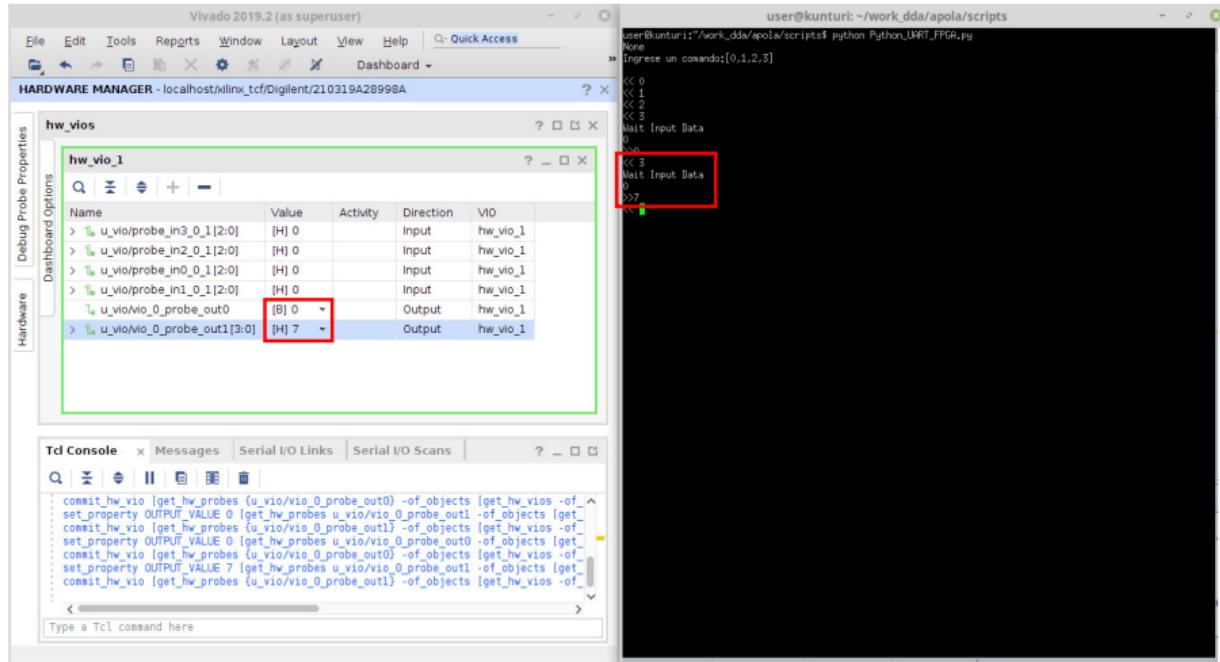
Elegimos 1 y activa el color Verde (2).

# VIO y Python



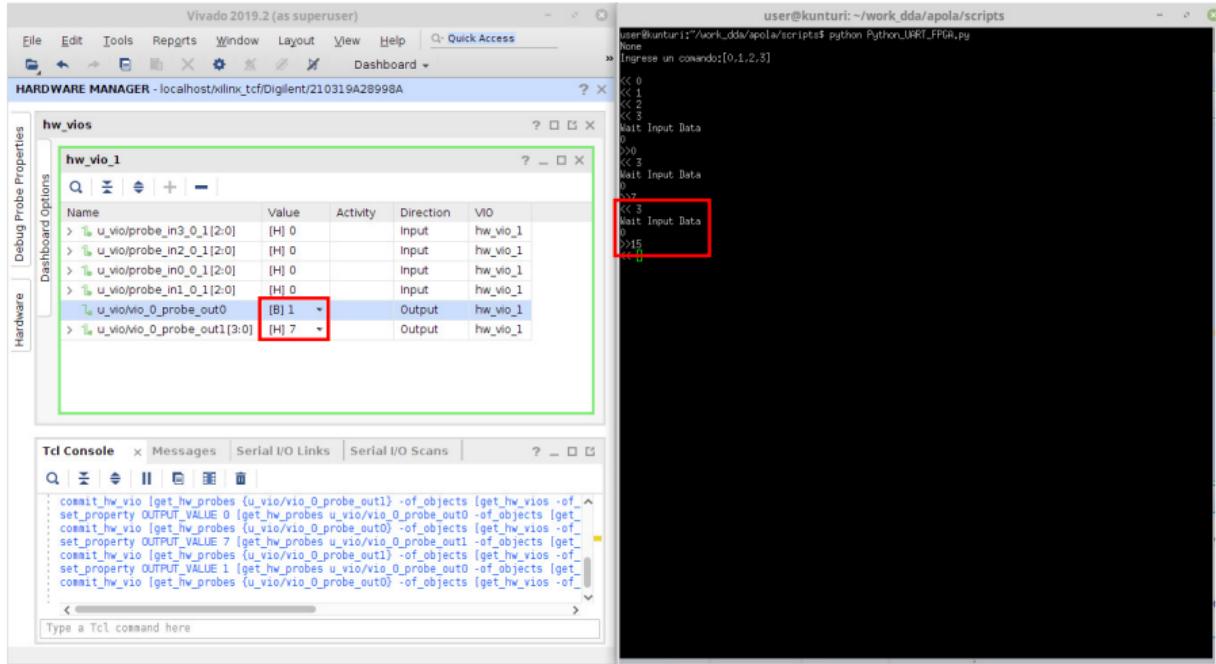
*Elegimos 2 y activa el color Azul (4).*

# VIO y Python



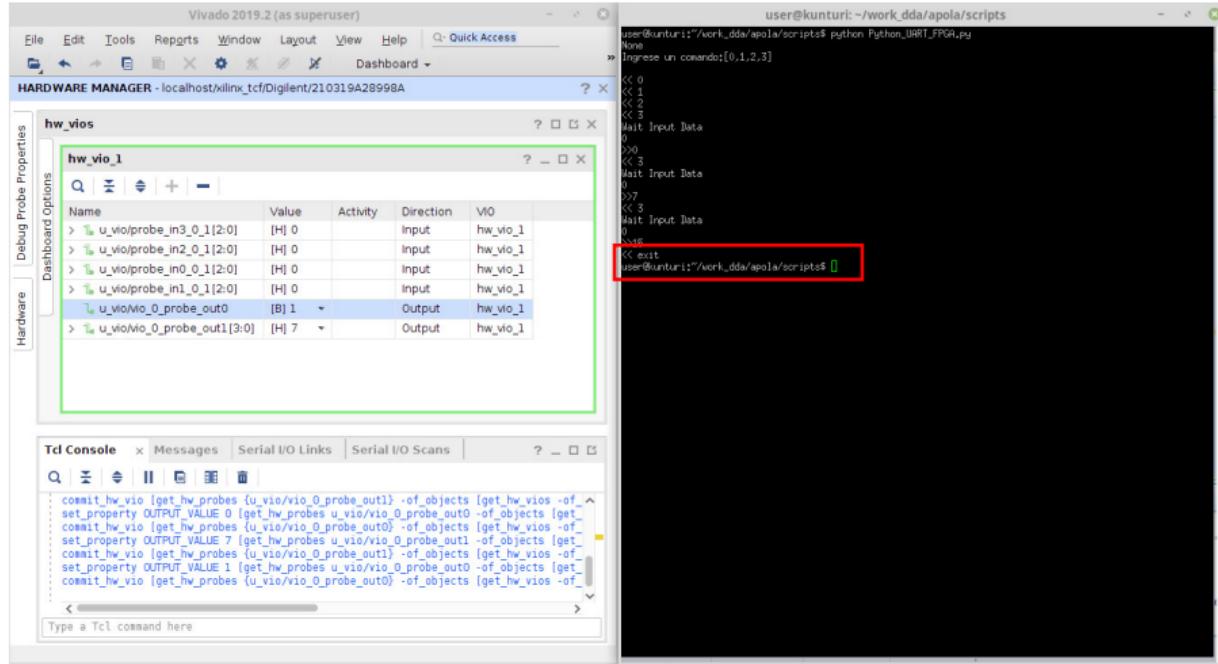
Elegimos 3 para leer el estado de los Switch. Con la salida out0: 0 habilitamos el control desde el VIO y el python tiene que leer el estado de la salida out1 (Python >> 0).

# VIO y Python



Elegimos 3 para leer el estado de los Switch. Con la salida out0: 0 habilitamos el control desde el VIO y el python tiene que leer el estado de la salida out1 (Python >> 7).

# VIO y Python



Elegimos 3 para leer el estado de los Switch. Con la salida out0: 1 habilitamos el control desde la FPGA y el python tiene que leer el estado de los switch (Python >> 15).

# Cierre de Sesiones

## Cerrando el Túnel

- Liberando el puerto USB
  - Para liberar el puerto USB asignado escribir la palabra **Exit**.
- Cerrando los dos túneles
  - Para cerrar ambas sesiones escribir la palabra **exit**.

The image displays two terminal windows side-by-side. Both windows show a command-line interface with a user prompt and some system information.

**Terminal Window 1 (Left):**

```
user@kunturi:~/work_dda/apola/scripts
* Support: https://ubuntu.com/advantage
Pueden actualizarse 27 paquetes.
0 actualizaciones son de seguridad.

Last login: Wed Nov 15 21:11:43 2017 from 200.126.231.123
user@kunturi:~$ cd work_dda/apola/scripts/
user@kunturi:~/work_dda/apola/scripts$ python client.py
-----
Write-> Exit <-to close the session
Ip: 127.0.0.1
-----
Port: 5005
-----
Checking Free Port
-----
USB Free: USB3-210319A2CECCA
-----
Write Text to check Session or Exit to close: Exit
Echo: Exit
Close Session
user@kunturi:~/work_dda/apola/scripts$ exit
```

**Terminal Window 2 (Right):**

```
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-38-generic x86_64)

* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage

Pueden actualizarse 27 paquetes.
0 actualizaciones son de seguridad.

Last login: Wed Nov 15 22:33:00 2017 from 200.126.231.123
user@kunturi:~$ cd work_dda/apola/scripts/
user@kunturi:~/work_dda/apola/scripts$ python protocolo_uart_dda.py 3
RESET ON
MODULES OFF
LOG RUN OFF
RESET OFF
ENB MODULES
LOG
LOG RUN OFF
LOG RUN ON
LOG READ
4096
End Script
user@kunturi:~/work_dda/apola/scripts$ exit
```

Las figuras muestran como liberar el USB asignado y el cierre de los túneles abiertos.