

## Diseño Digital Avanzado

### Microblaze - GPIO - VIO

Dr. Ariel L. Pola

[ariel.pola@mi.unc.edu.ar](mailto:ariel.pola@mi.unc.edu.ar)

December 14, 2024

# 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. Tunel, Programar la FPGA y Ejecutar la Aplicación en Forma Remota

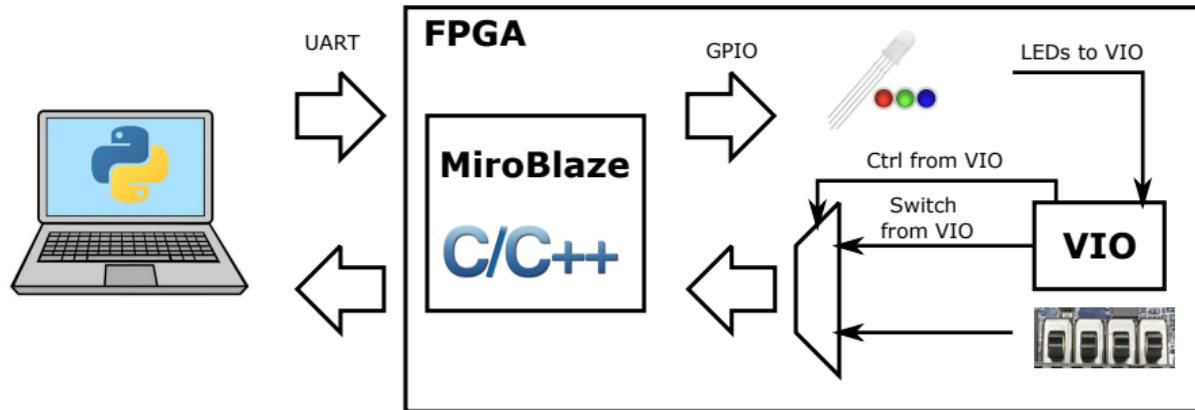
# Introducción



# Introducción

- El objetivo de esta presentación es detallar paso a paso la instalación y programación de un micro-embobido.
- 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.

# Introducción

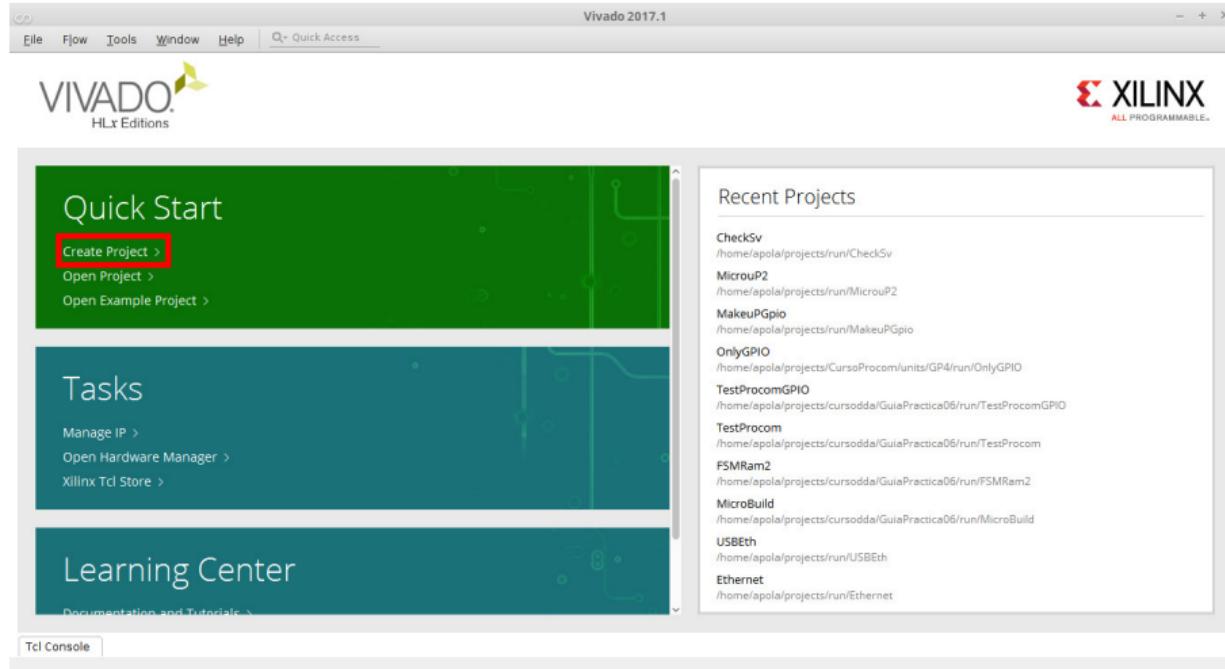


*Descripción general.*

# Nuevo Proyecto

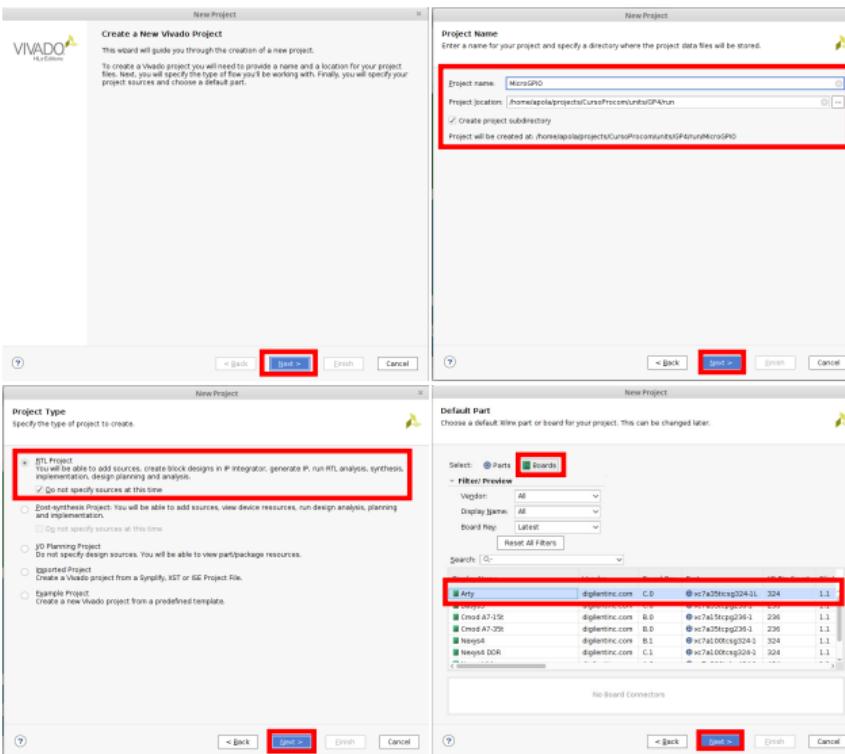


# Nuevo Proyecto



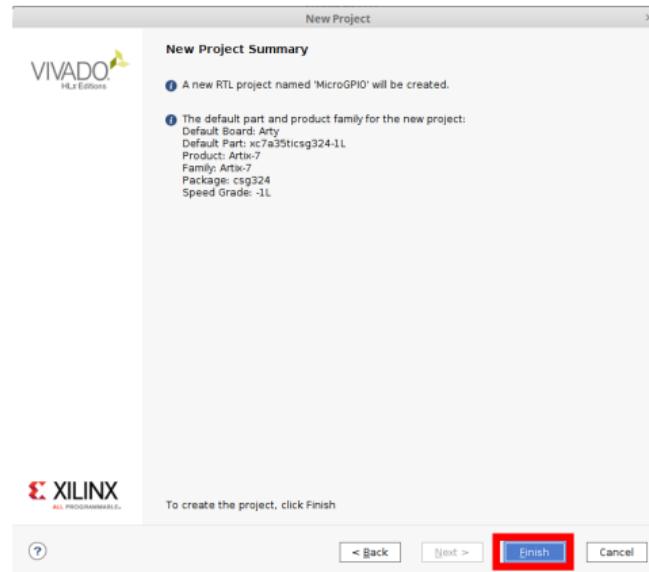
*Crear un nuevo proyecto.*

# Nuevo Proyecto



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

# Nuevo Proyecto

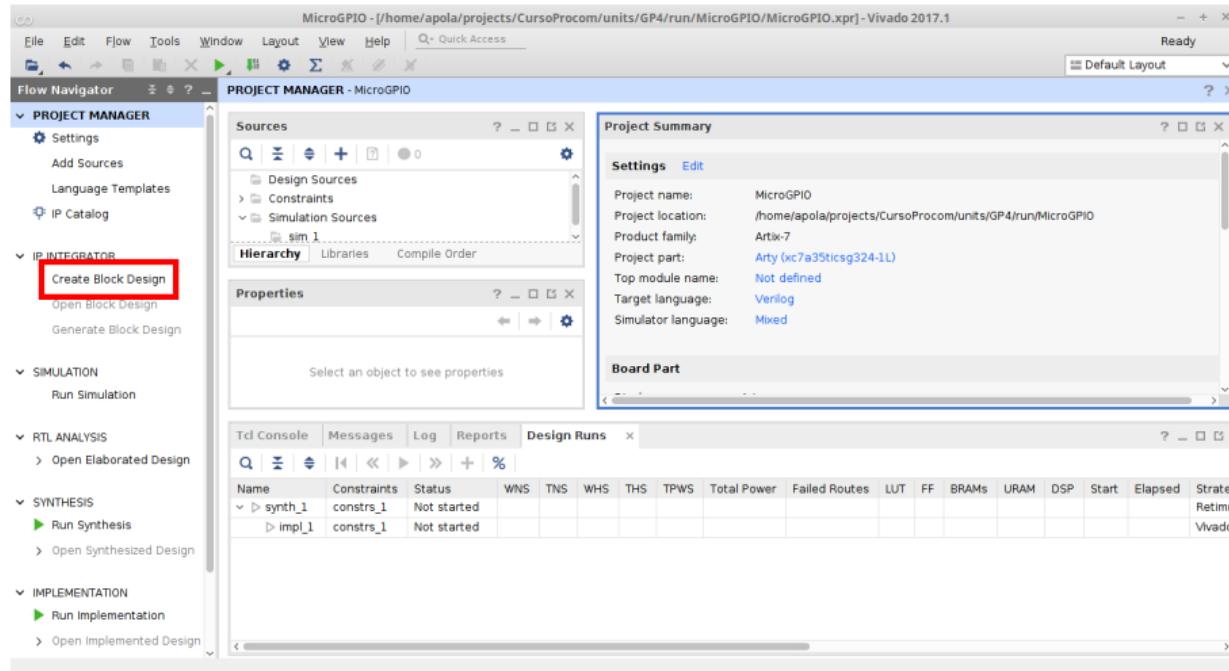


*Resumen del proyecto.*

## Creando un nuevo diseño

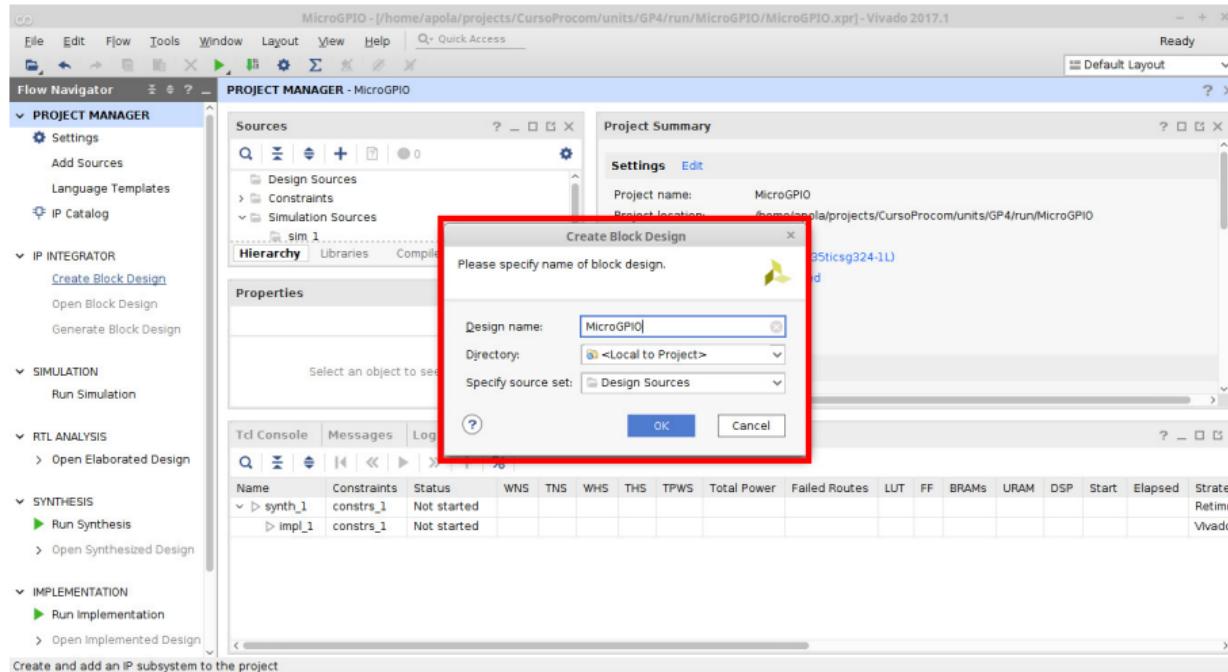


# Creando un nuevo diseño



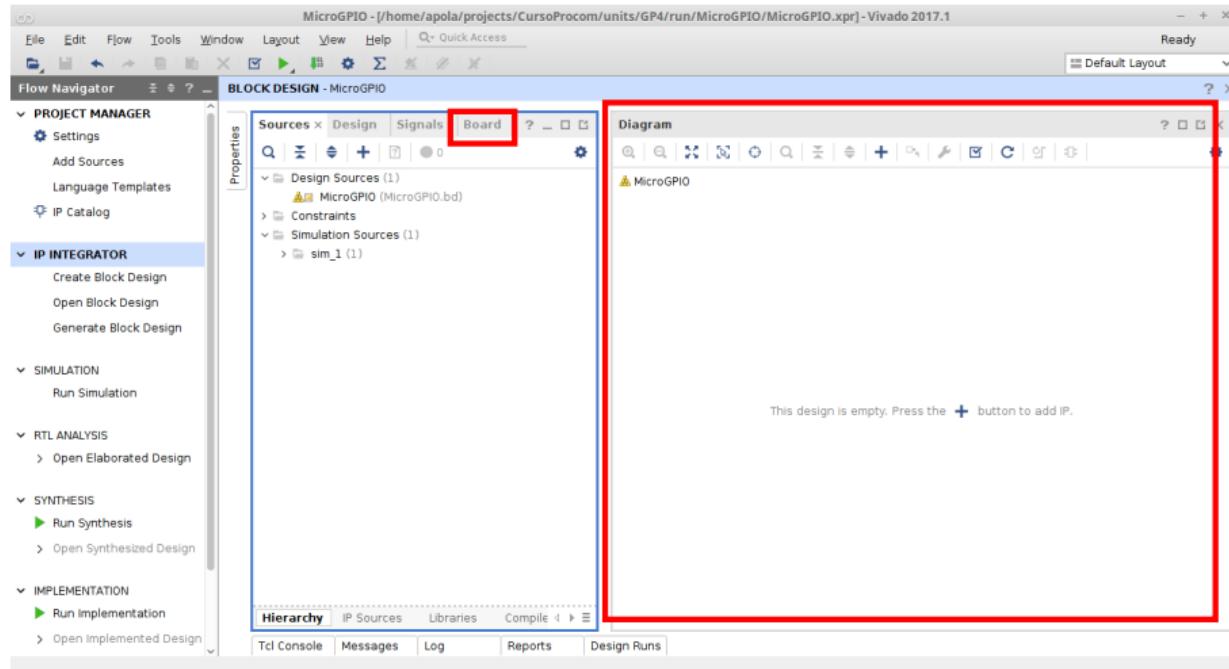
*Crean un nuevo bloque.*

# Creando un nuevo diseño



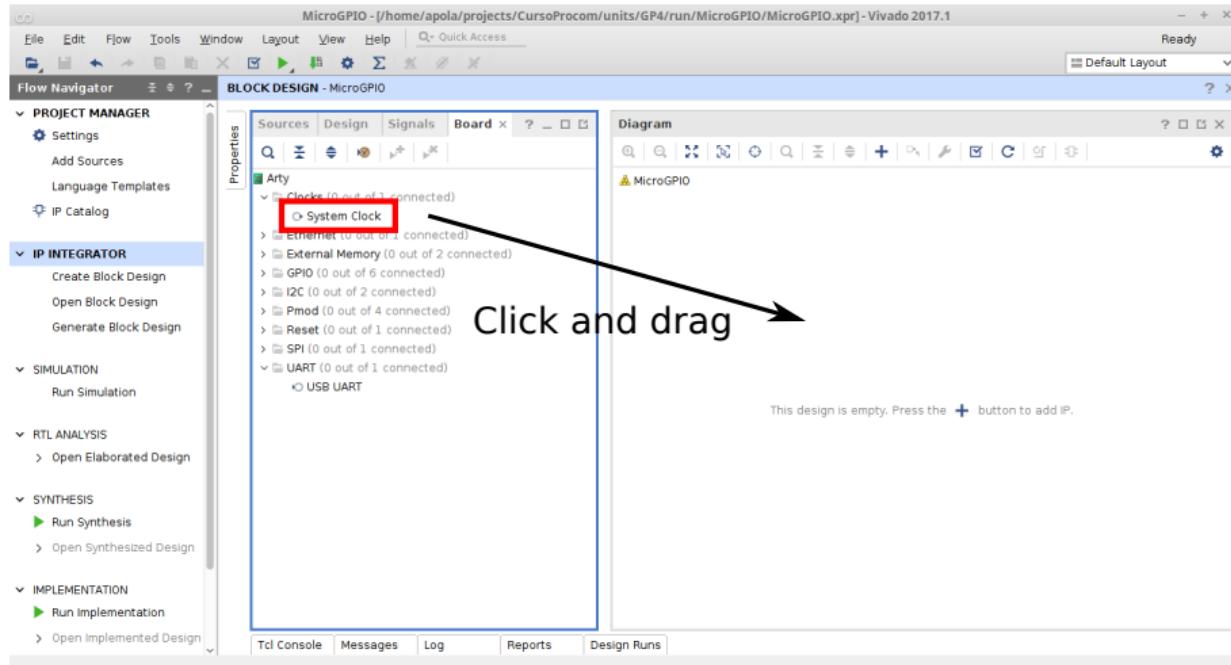
*Definir el nombre del nuevo diseño y el directorio de trabajo.*

# Creando un nuevo diseño



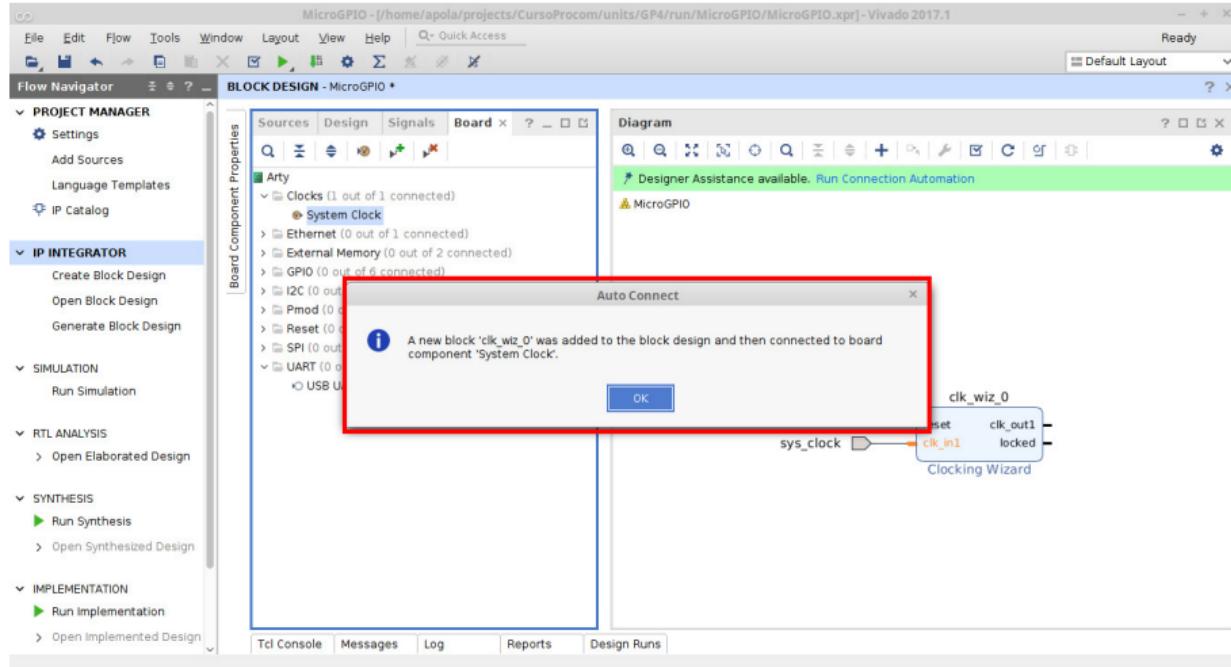
Nueva ventana de trabajo denominada “Diagrama” y en la pestaña “Board” se definen todos los componentes de la FPGA seleccionada.

# Creando un nuevo diseño



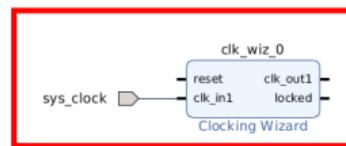
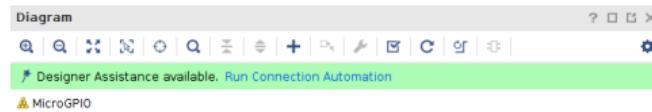
Instanciar “System Clock” haciendo click y arrastrando hacia la ventana.

# Creando un nuevo diseño



Mensaje de instancia de bloque.

# Creando un nuevo diseño

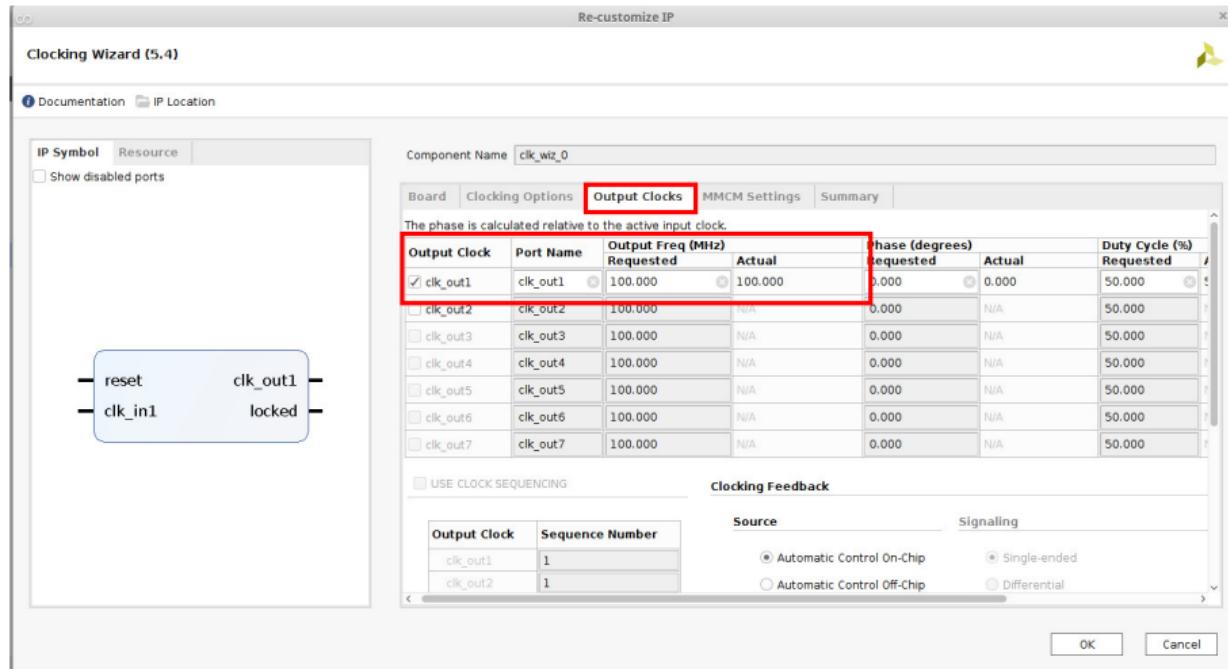


Double Click

---

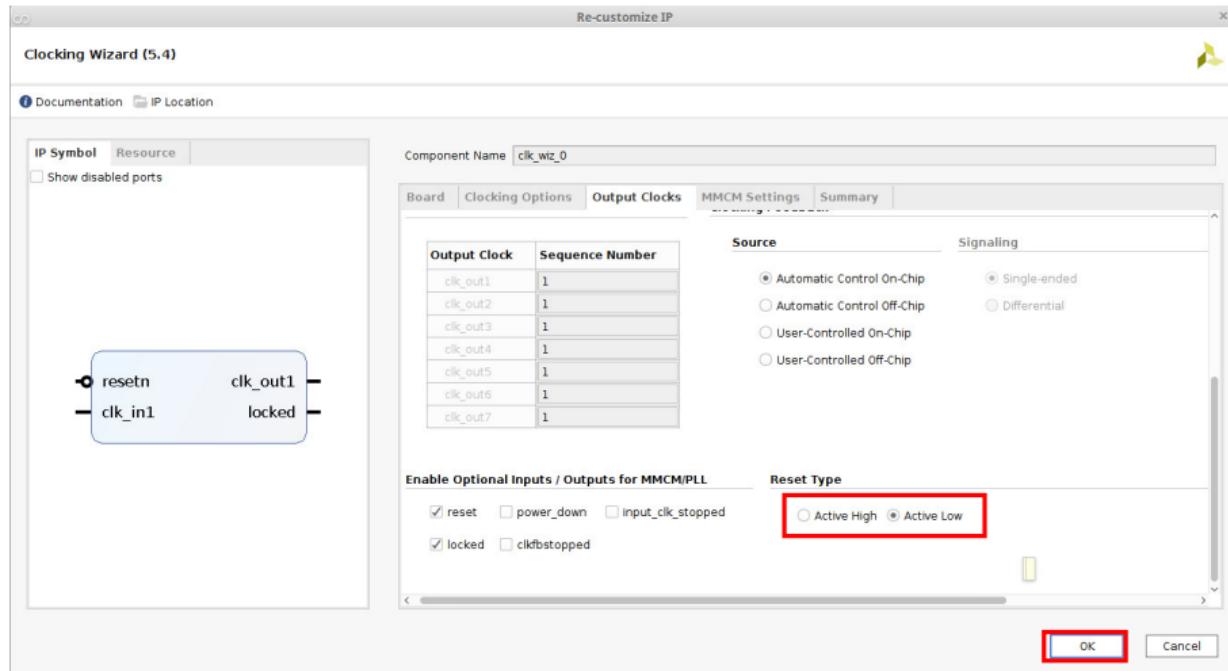
*Hacer doble click sobre “Clocking Wizard”.*

# Creando un nuevo diseño



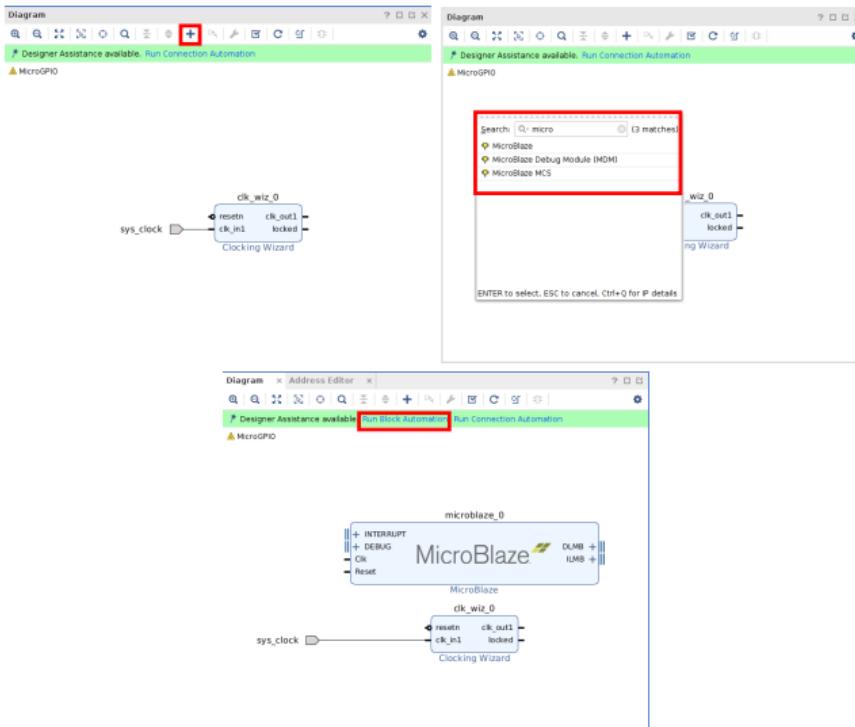
En la pestaña “Output Clocks” asignar la frecuencia de salida en 100MHz.

# Creando un nuevo diseño



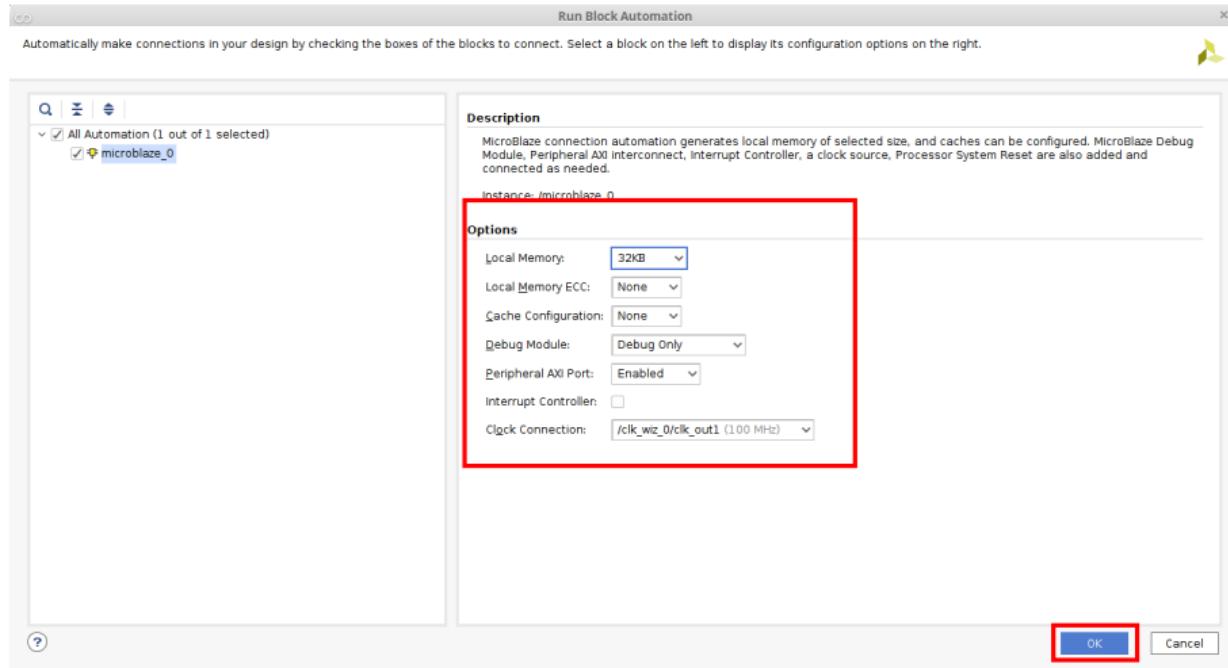
*El reset del bloque tiene que ser activo por bajo.*

# Creando un nuevo diseño



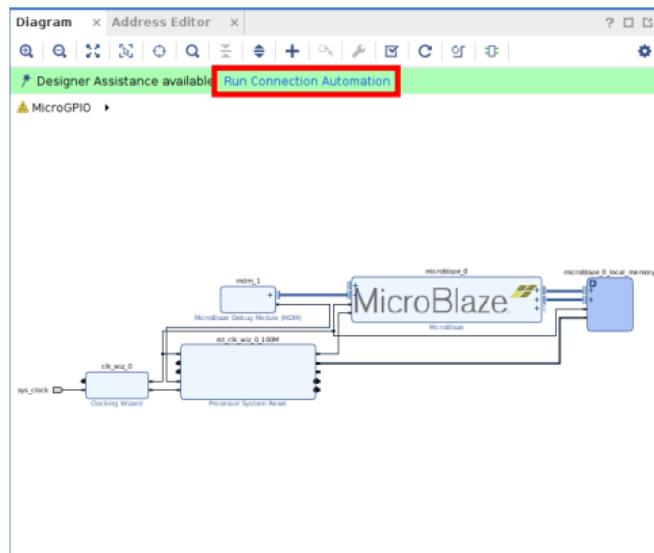
Incluir el core del MicroBlaze y ejecutar “Run Block Automation”.

# Creando un nuevo diseño



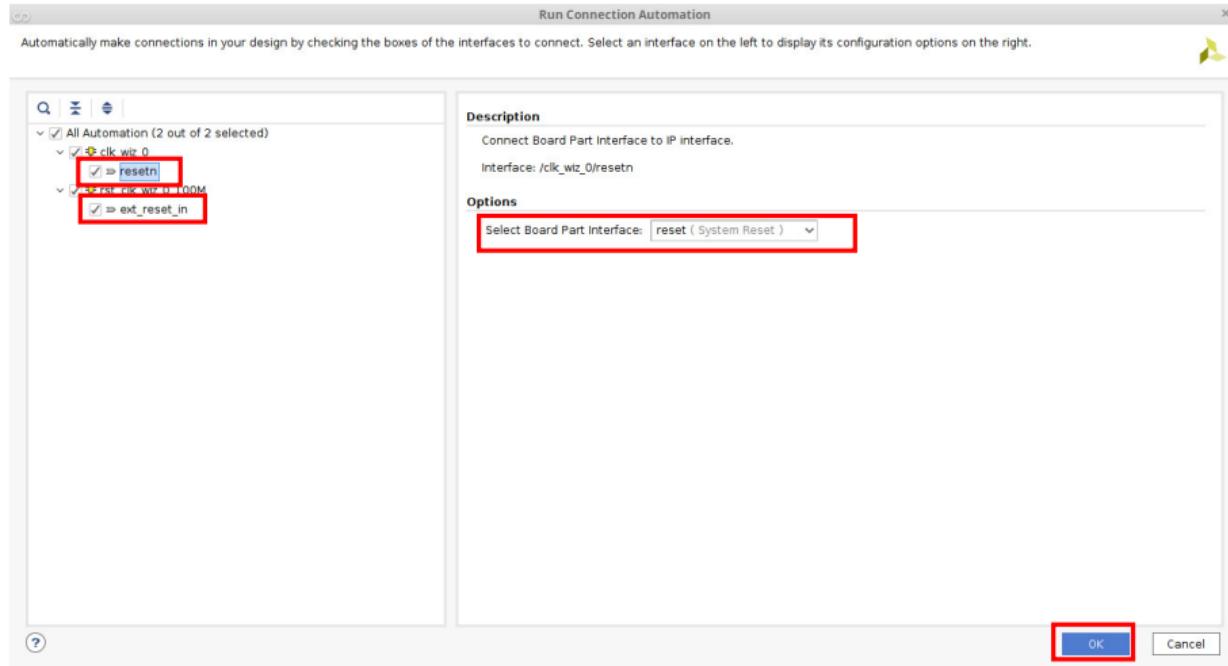
*Signar los parámetros de diseño del micro.*

# Creando un nuevo diseño



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

# Creando un nuevo diseño

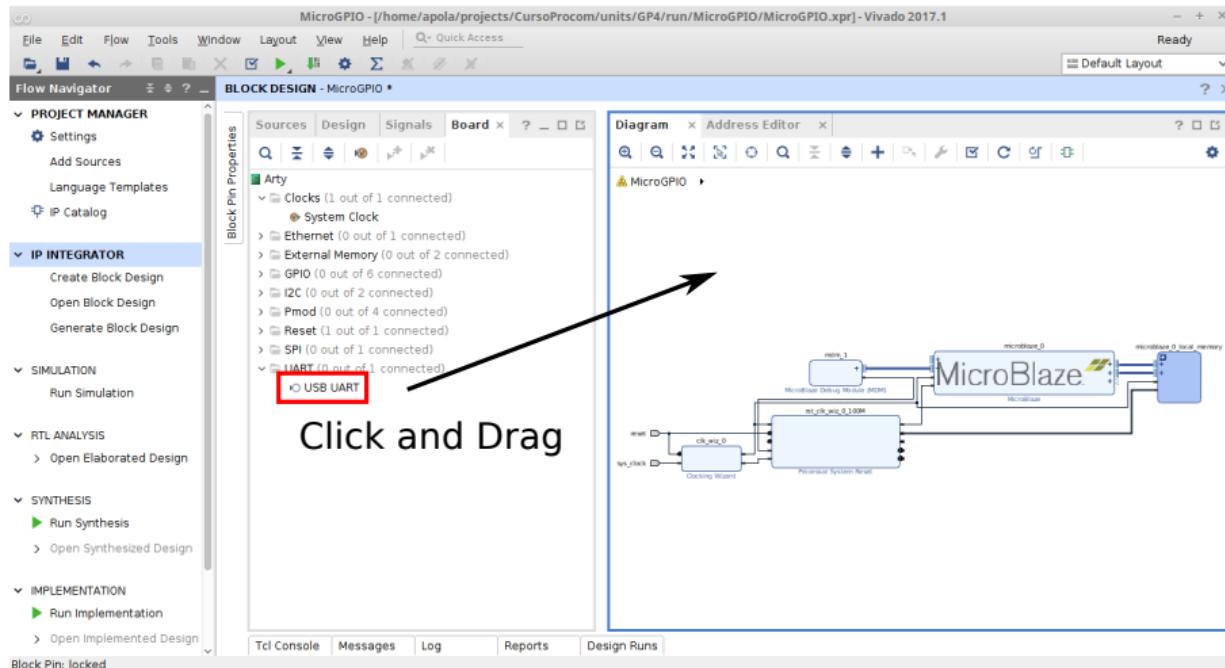


Verificar que los dos puertos de reset esten conectados al reset del sistema.

## Instancias de periféricos

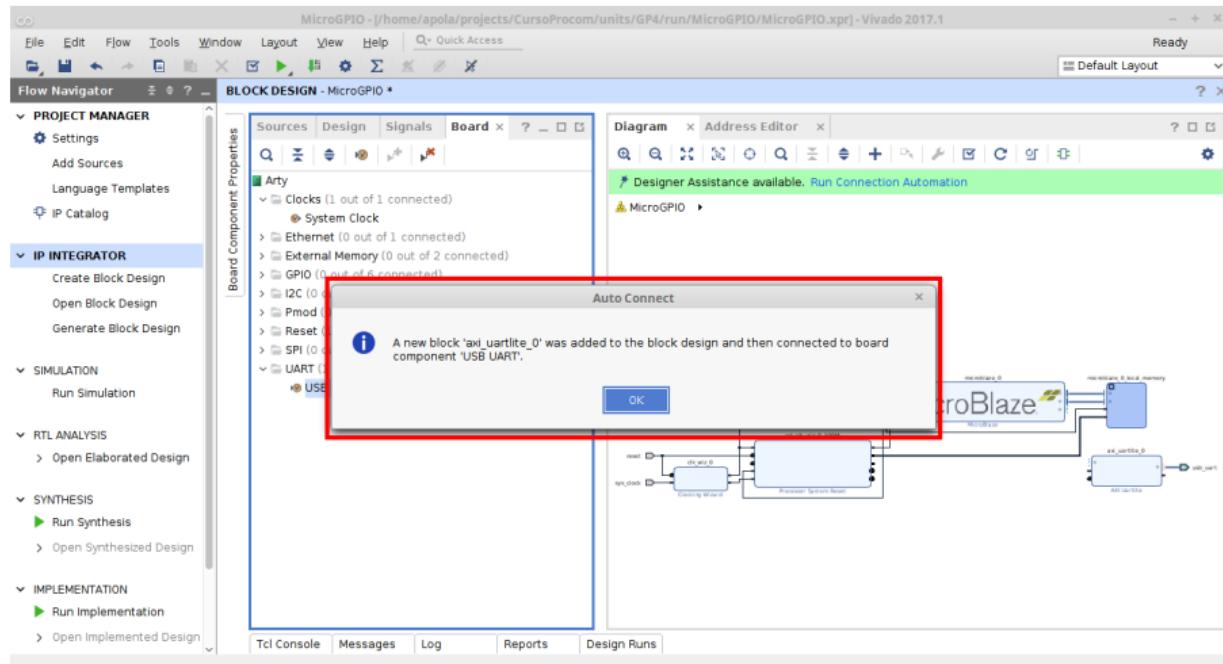


# Instancias de periféricos



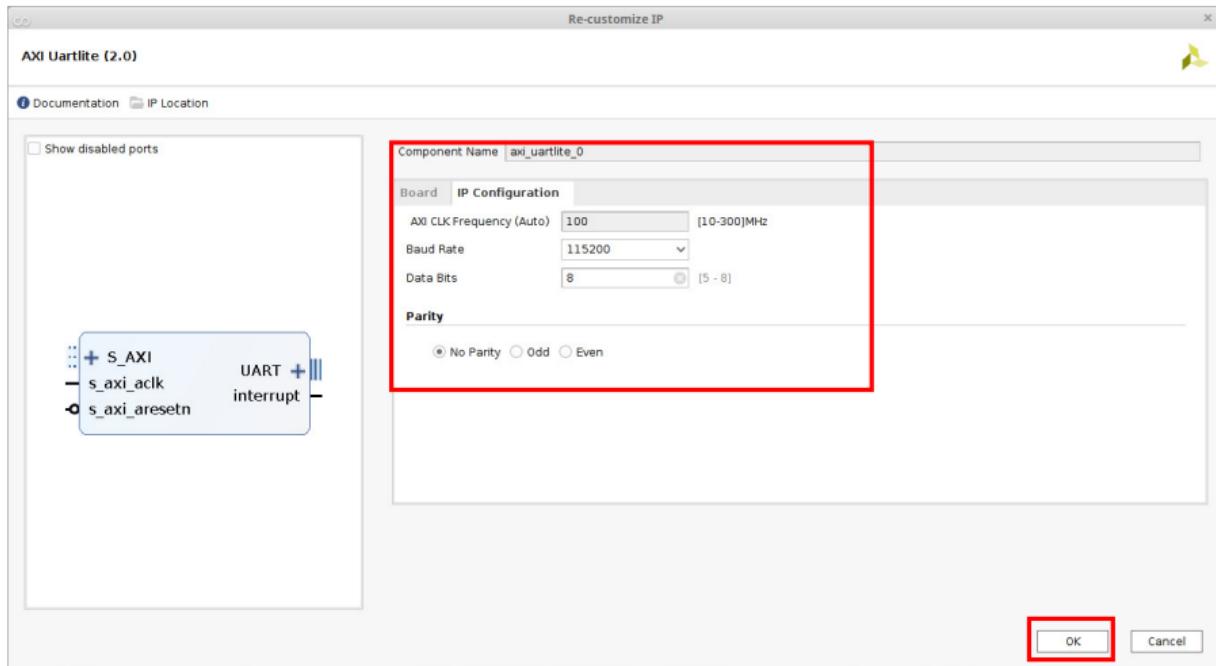
Instanciar el IP USB UART.

# Instancias de periféricos



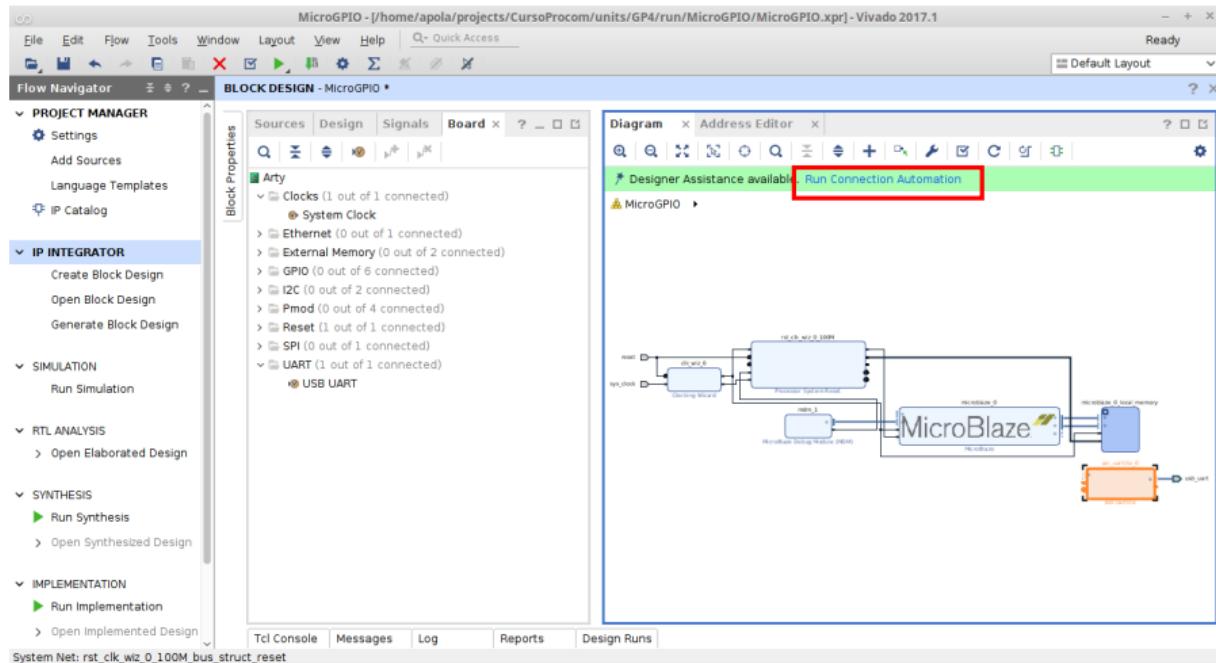
Verificar la instancia.

# Instancias de periféricos



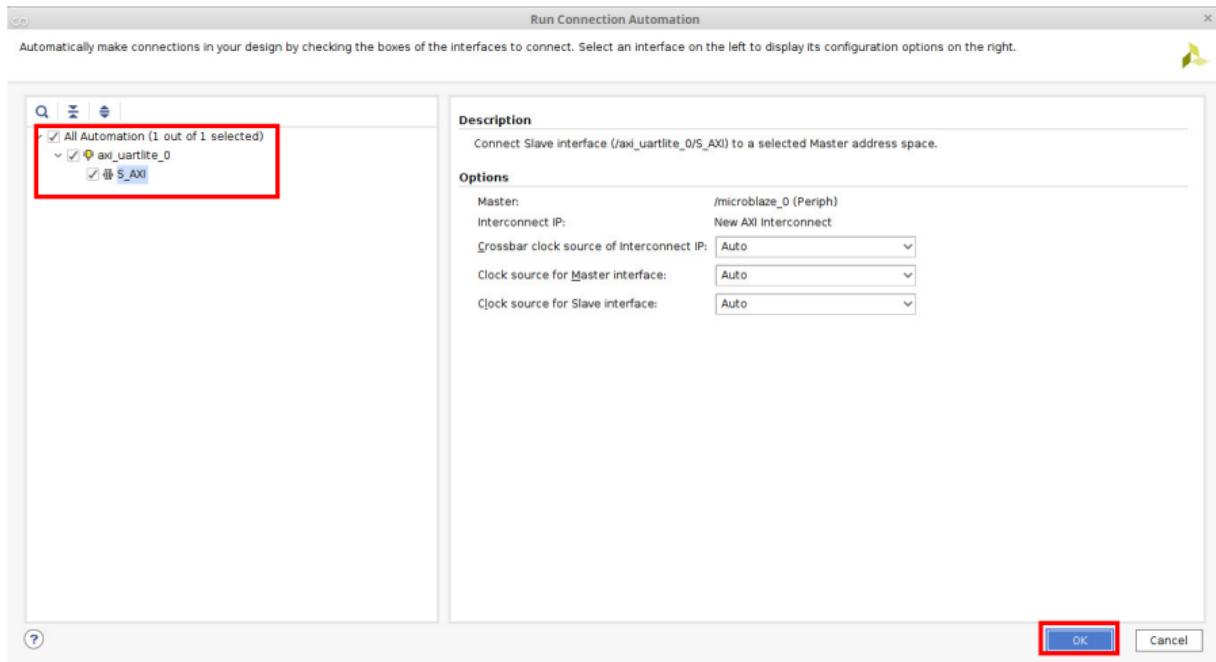
Hacer doble click sobre el periférico **UART** y configurar el **Baud Rate** en 115200.

# Instancias de periféricos



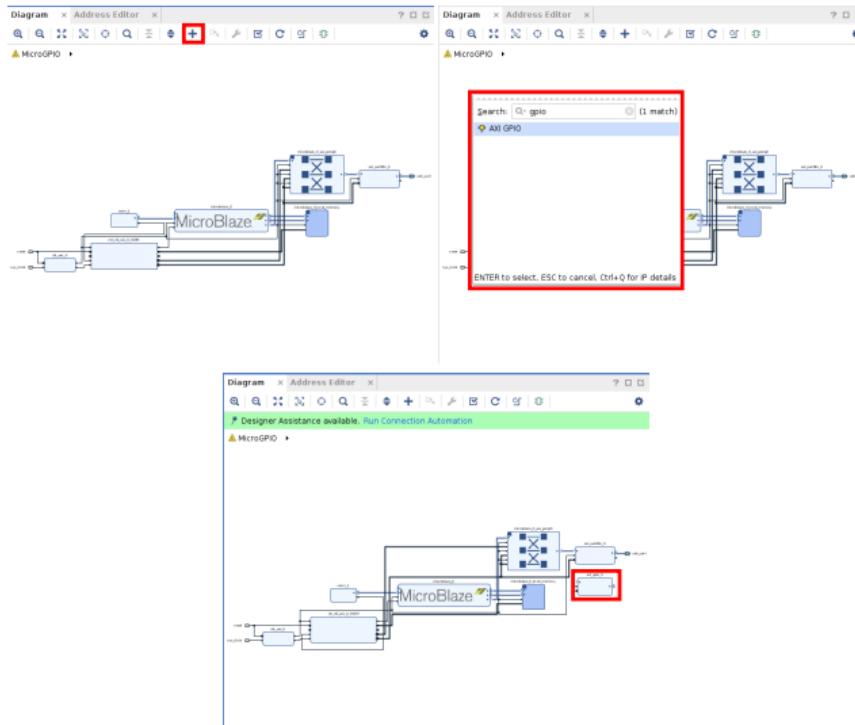
Seleccionar la conexión automática.

# Instancias de periféricos



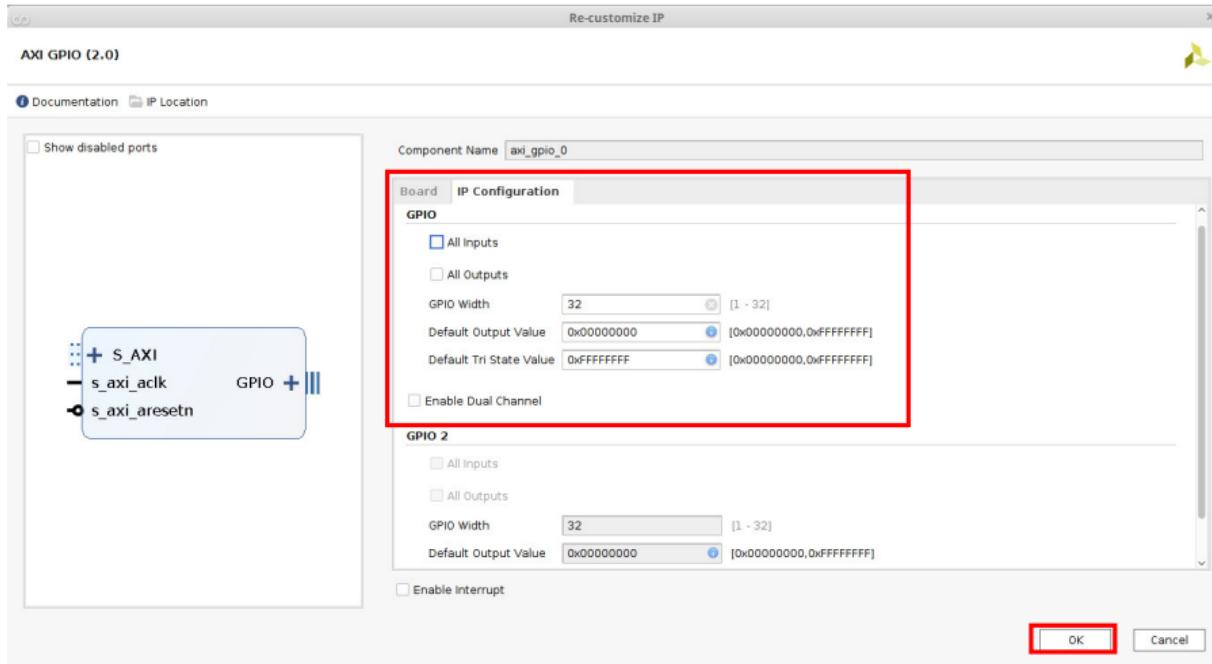
*Verificar que todos los puertos estén seleccionados.*

# Instancias de periféricos



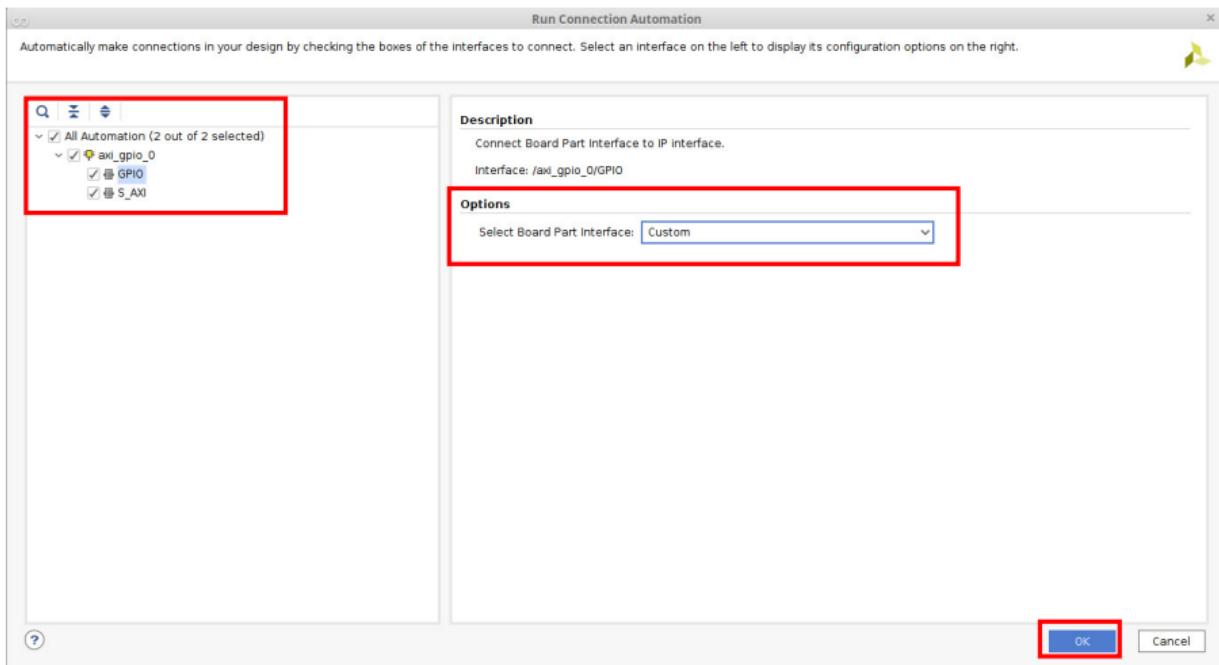
*Agregar un nuevo componente GPIO y hacer doble click sobre el IP.*

# Instancias de periféricos



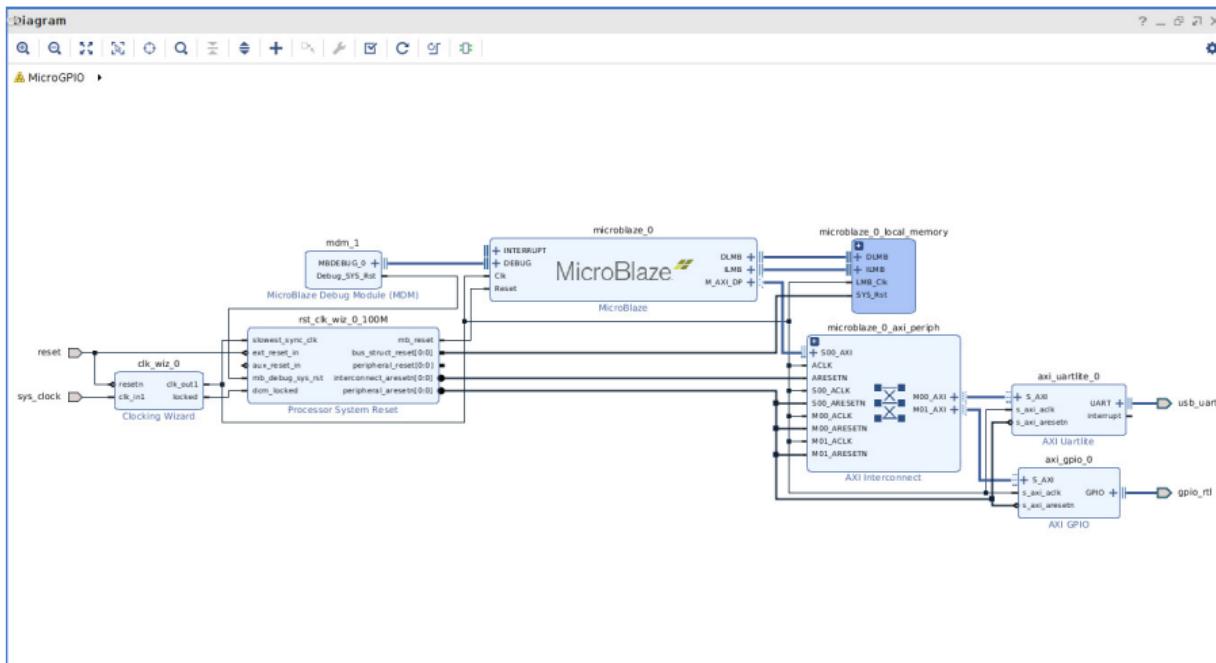
Verificar la configuración del puerto.

# Instancias de periféricos



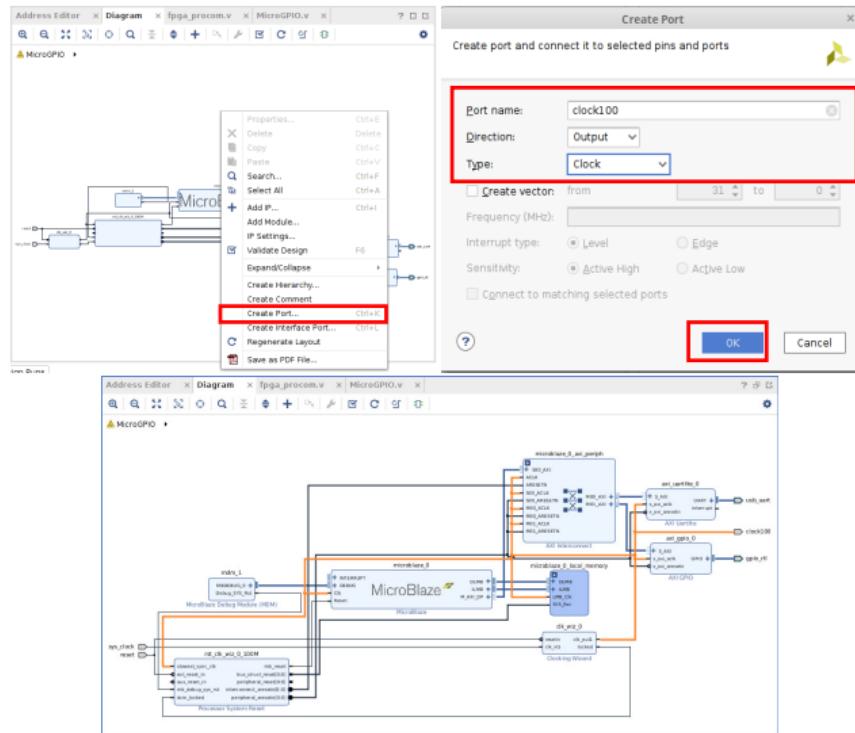
Seleccionar "Run Connection Automation" y verificar que el puerto de salida sea "Custom".

# Instancias de periféricos



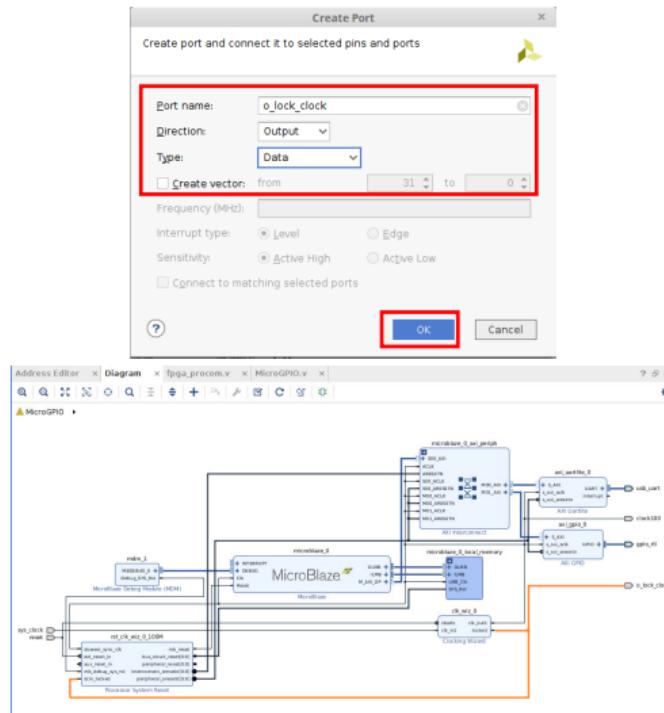
Diseño preliminar.

# Instancias de periféricos



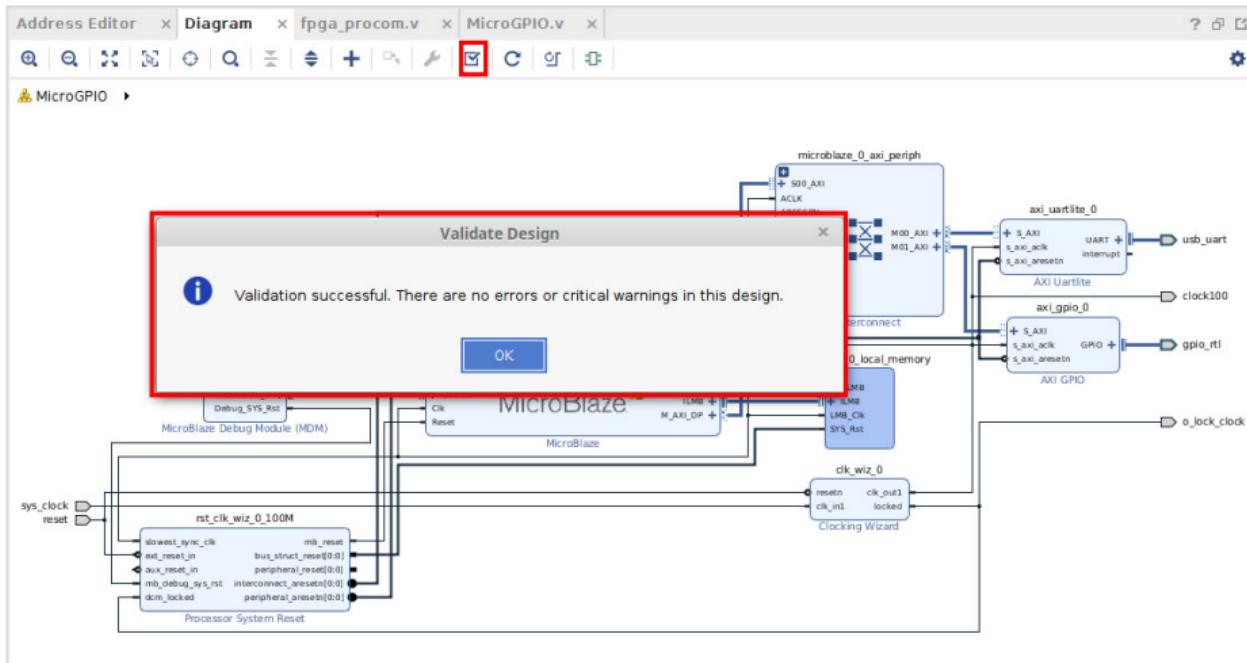
Incluir un nuevo puerto de clock de salida.

# Instancias de periféricos



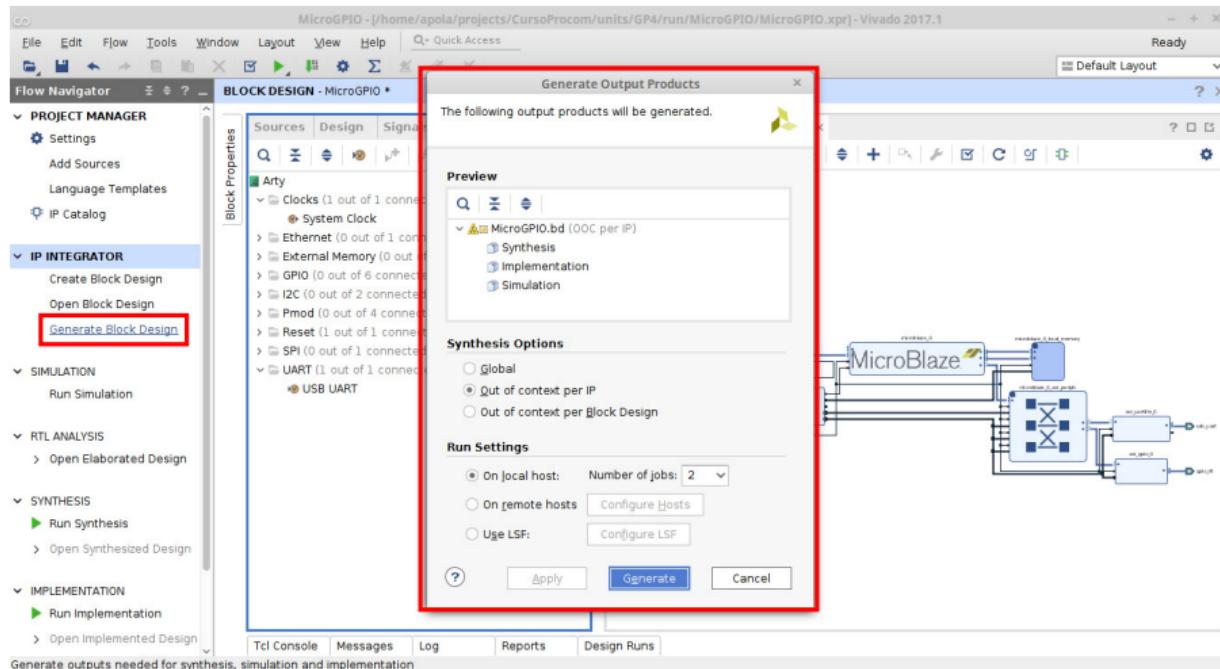
Repitir el paso anterior incluyendo un nuevo puerto de lock de clock.

# Instancias de periféricos



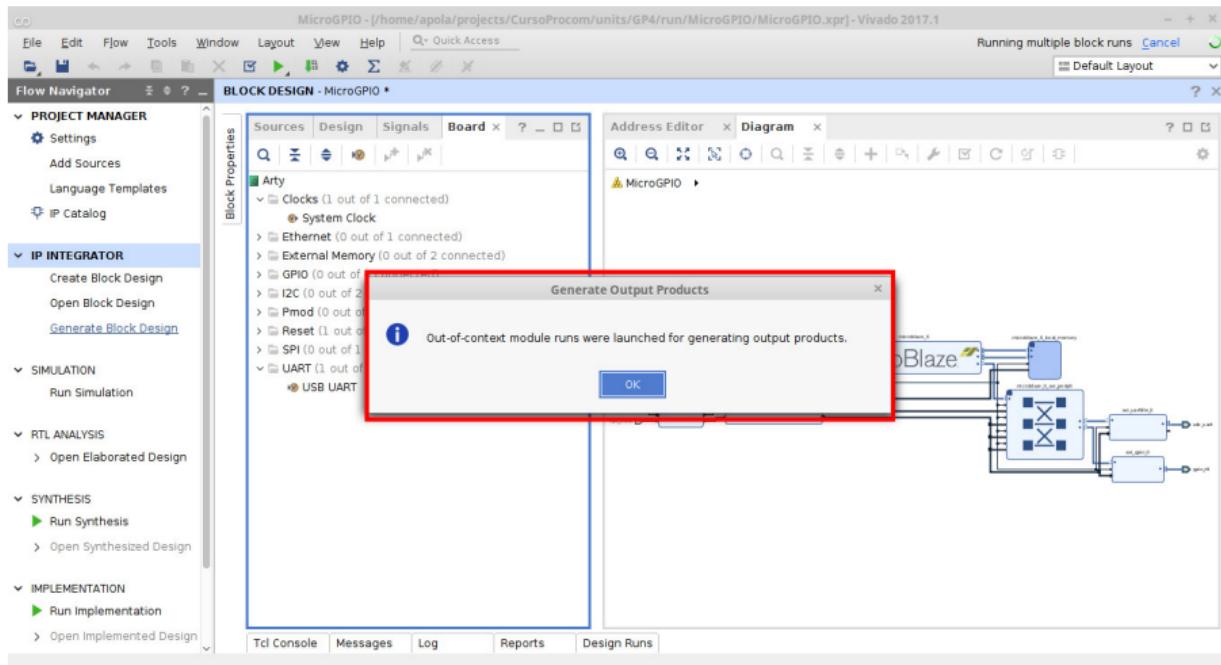
Validar el diseño.

# Instancias de periféricos



Generar el diseño completo.

## Instancias de periféricos

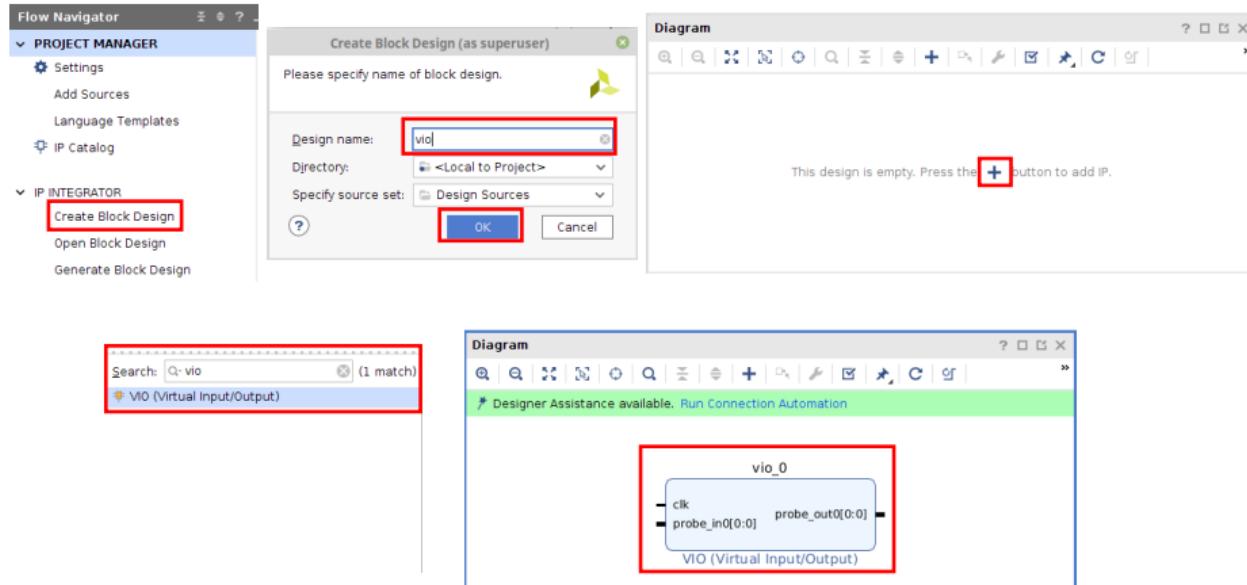


*Verificar la elaboración.*

## Instancia de VIO

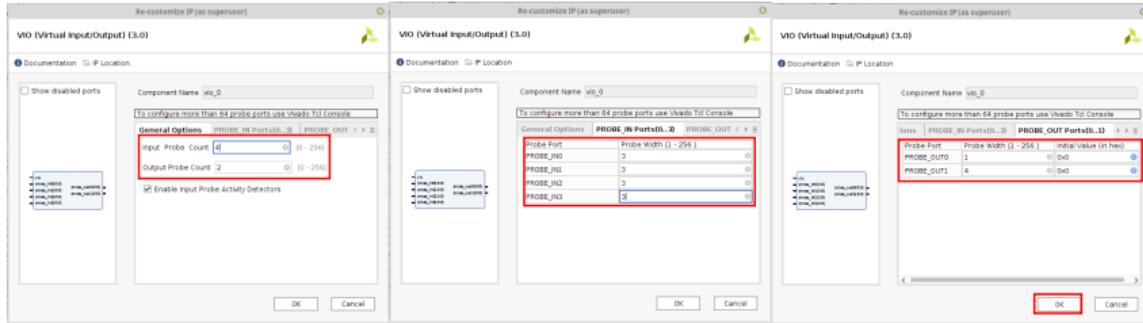


# Instancia de VIO



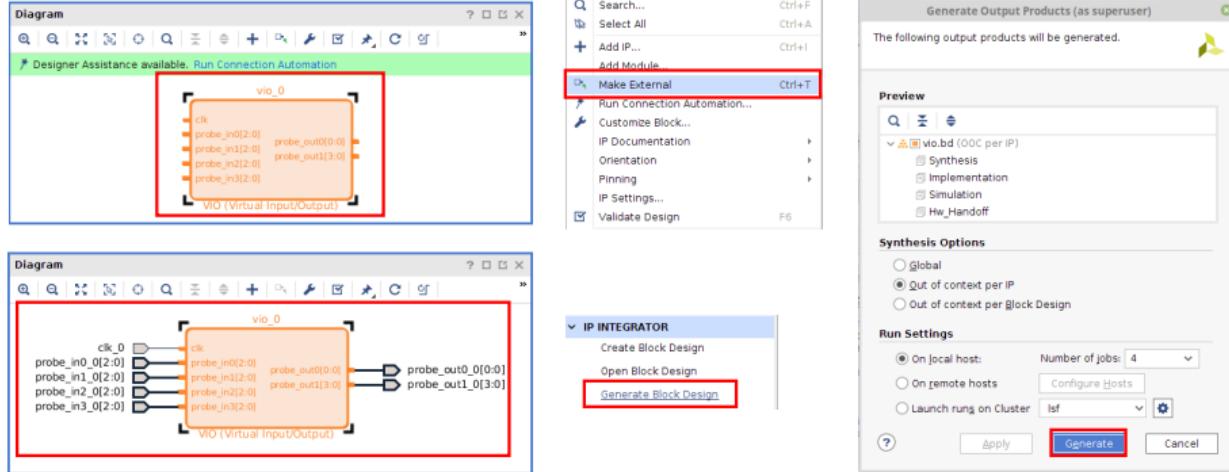
En caso de necesitar controlar en forma remota la FPGA debemos incluir el bloque VIO a nuestro diseño. Creamos un nuevo bloque, le asignamos el nombre, agregamos un nuevo IP, seleccionamos el IP VIO y hacemos doble click sobre el IP generado.

# 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

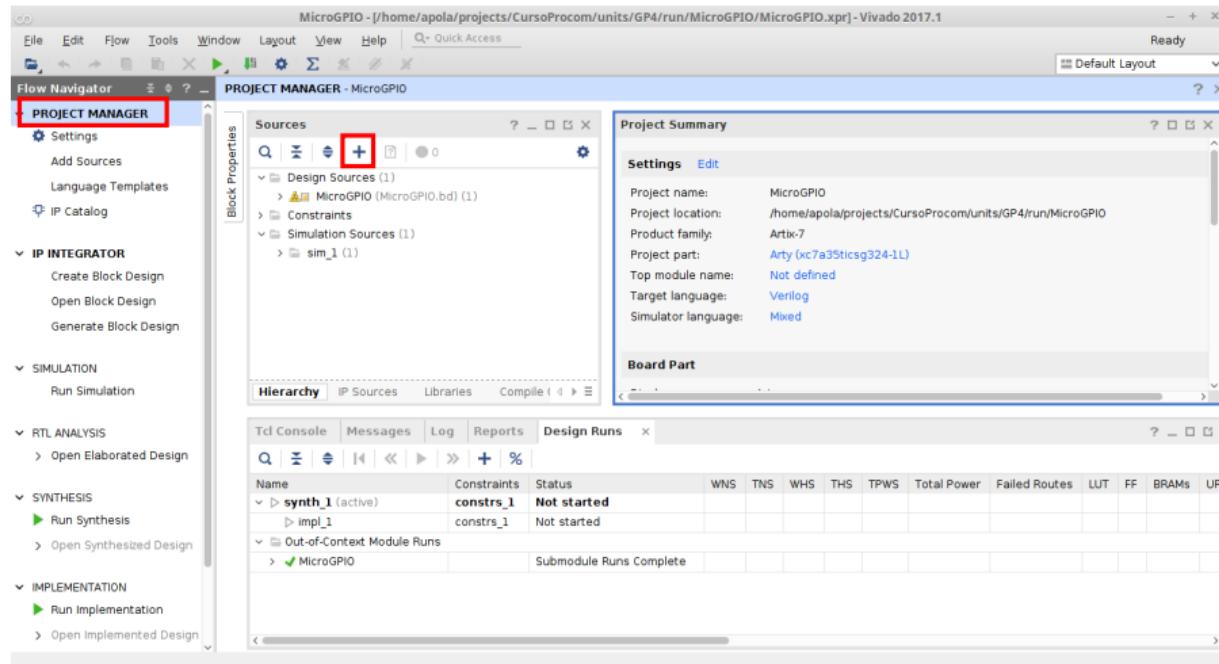


Sobre el bloque modificado hacemos click con el botón derecho, seleccionamos Make External lo cual agrega los puertos exteriores y por último generamos el bloque.

## Instanciar el uP en Top Level

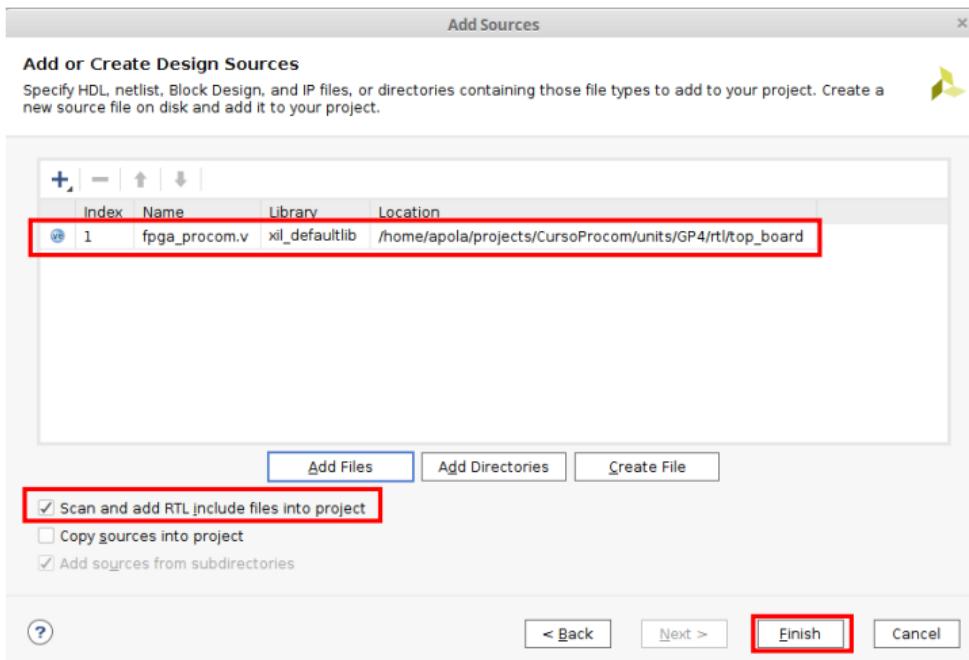


# Instanciar el uP en Top Level



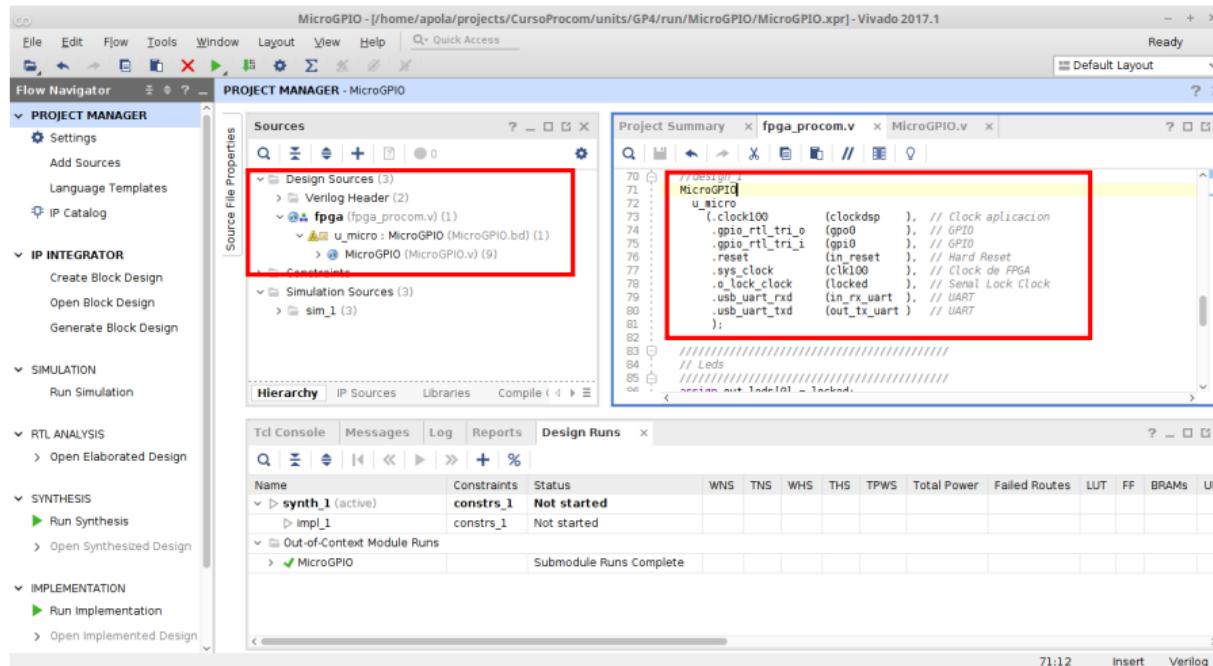
*Agregar un nuevo archivo verilog.*

# Instanciar el uP en Top Level



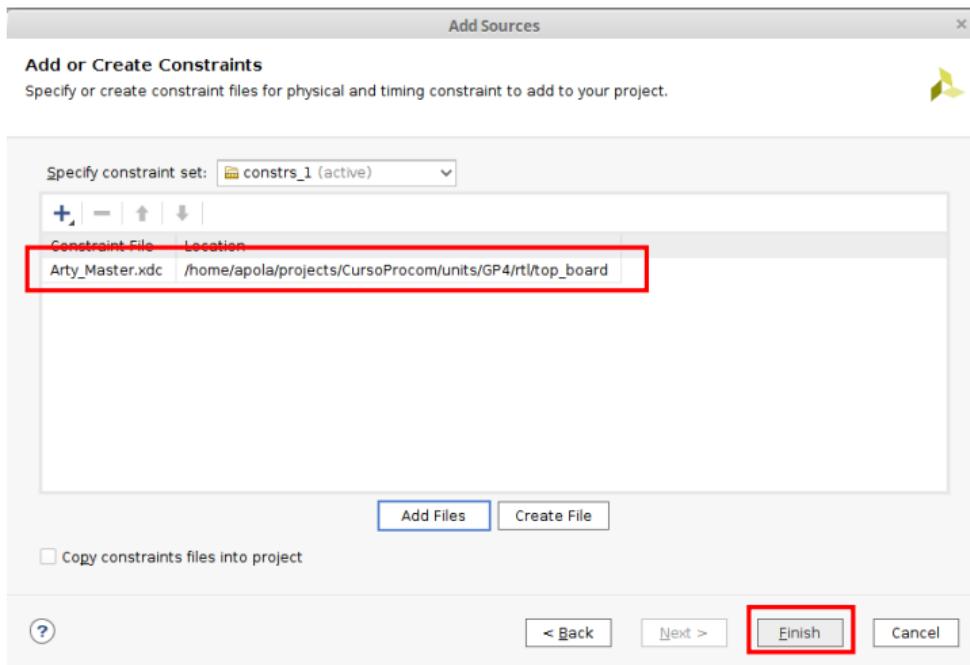
Seleccionar el archivo “fpga\_dda” y asegurarse de activar el escaneo de includes.

# Instanciar el uP en Top Level



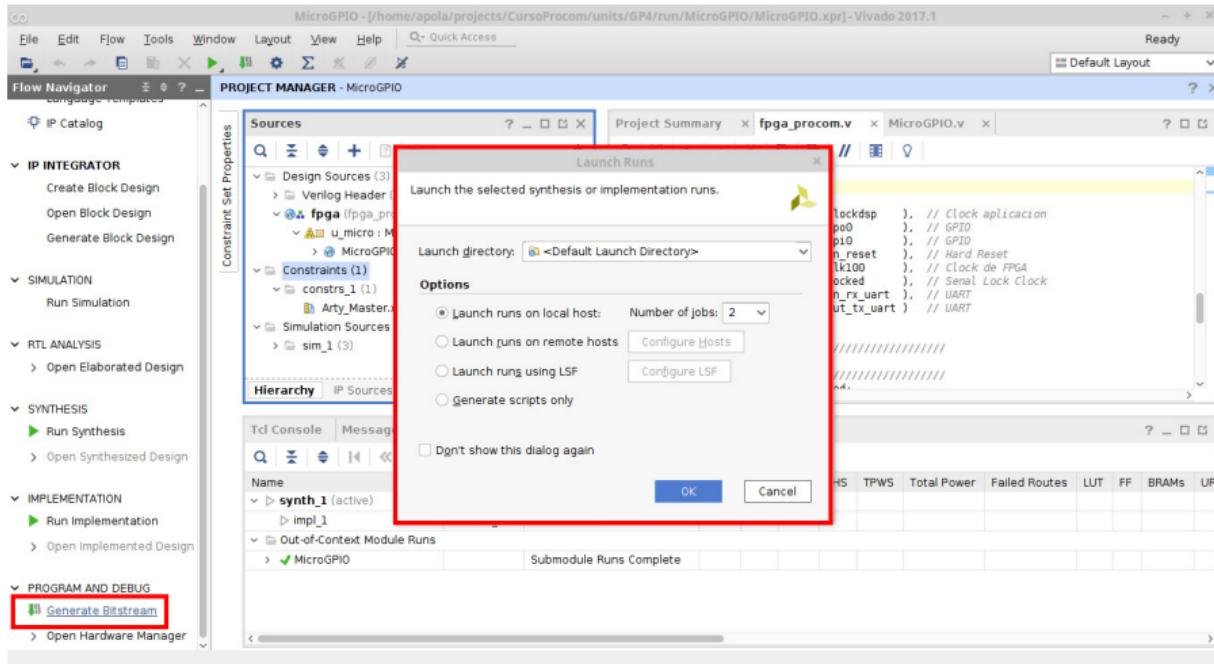
Verificar que el nombre del archivo del micro sea el mismo que se utiliza en el top level y los mismos puertos.

# Instanciar el uP en Top Level



*Agregar el archivo de puertos físicos.*

# Instanciar el uP en Top Level



Generar el binario.

## 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.



The screenshot shows the Vivado IDE interface. On the left, the 'Design Sources' panel lists the following files:

- fpga (fpga\_procom.v) (1)
- vio (vio.bd) (1)
  - vio (vio.v) (1)
- Constraints (1)
- Simulation Sources (2)
- Utility Sources

The main window displays the Verilog code for 'fpga\_procom.v'. The code defines two wires of type [NB\_GPIOS-1:0] named 'gpo0' and 'gpio', both initialized to 0. It also declares a wire 'locked' and three wires 'soft\_reset', 'clockSel', and 'fromHard'. A fourth wire, 'fromVio', is declared but not yet assigned a value. The code is annotated with comments // Vars and //.

```
45 //////////////////////////////////////////////////////////////////
46 // Vars
47 //////////////////////////////////////////////////////////////////
48 wire [NB_GPIOS-1:0] gpo0;
49 wire [NB_GPIOS-1:0] gpio;
50
51 wire locked;
52
53 wire soft_reset;
54 wire clockSel;
55 wire fromHard;
56 wire [3 : 0] 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 displays the Xilinx Vivado IDE interface with three main windows:

- Sources:** Shows the project structure with "fpga (fpga\_procom.v) (1)" and "vio (vio.v) (1)" listed under "Design Sources".
- Diagram:** Shows the Verilog code for the "vio.v" module. The code includes a module definition for "vio" with various probe and in/out ports, and a instantiation of the "Integrator" IP core.
- Hierarchy:** Shows the project structure with "fpga (fpga\_procom.v) (1)" and "vio (vio.v) (1)" listed under "Design Sources".

Red boxes highlight the "vio" module instances in the Sources/Hierarchy windows and the "vio" module declaration in the "vio.v" code.

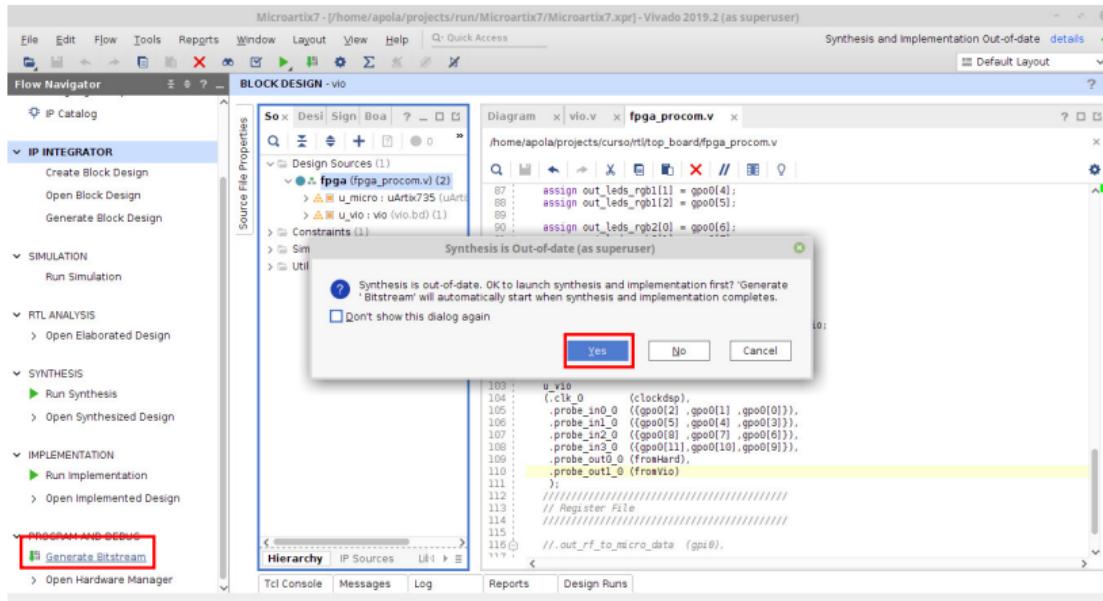
```
Diagram : vio.v
timescale 1 ps / 1 ps
module vio
    // CORE GENERATION INFO: v5.00 Integrator,{x_ipVendor=xilinx.com,x_ipLibrary=}
    input [1:0] i_clk;
    output [1:0] o_out;
    probe_in0_0;
    probe_in1_0;
    probe_in2_0;
    probe_in3_0;
    probe_out0_0;
    probe_out1_0;
    (* X_INTERFACE_INFO = "xilinx.com:signal:clock:1 CLK(CLK_0 CLK# *)")
    input [2:0]probe_in0_0;
    input [2:0]probe_in1_0;
    input [2:0]probe_in2_0;
    input [2:0]probe_in3_0;
endmodule
```

```
Diagram : fpga_procom.v
assign out_leds_rgb1[1] = gpo0[4];
assign out_leds_rgb1[2] = gpo0[5];
assign out_leds_rgb2[0] = gpo0[6];
assign out_leds_rgb2[1] = gpo0[7];
assign out_leds_rgb2[2] = gpo0[8];
assign out_leds_rgb3[0] = gpo0[9];
assign out_leds_rgb3[1] = gpo0[10];
assign out_leds_rgb3[2] = gpo0[11];
assign gpo0[3 : 0] = (fromHard) ? i_sw : fromVio;
//assign gpo0[3 : 0] = i_sw;
assign gpo1[0:1] = (2'b11 0:1);
vio
    u_vio
        (.clk_0      (clockdsp),
         .probe_in0_0 ((gpo0[2].gpo0[1].gpo0[0])),
         .probe_in1_0 ((gpo0[5].gpo0[4].gpo0[3])),
         .probe_in2_0 ((gpo0[8].gpo0[7].gpo0[6])),
         .probe_in3_0 ((gpo0[11].gpo0[10].gpo0[9])),
         .probe_out0_0 (fromHard),
         .probe_out1_0 (fromVio)
```

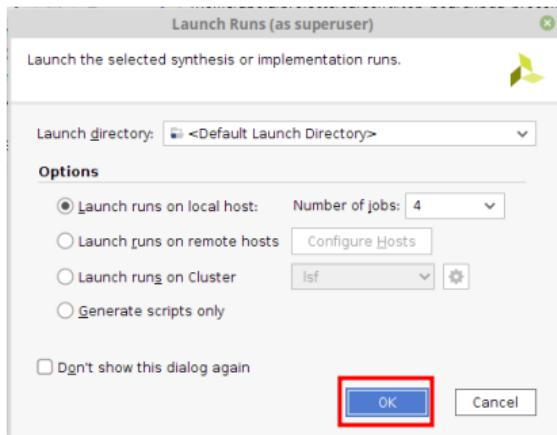
Además, debemos copiar la declaración del módulo VIO e incluirlo en el Top Level. Por último, asignamos la selección del control por medio del bloque VIO y comentamos la asignación directa de los switch.

# Instanciar el VIO en Top Level



*Generamos el binario nuevamente.*

# Instanciar el VIO en Top Level

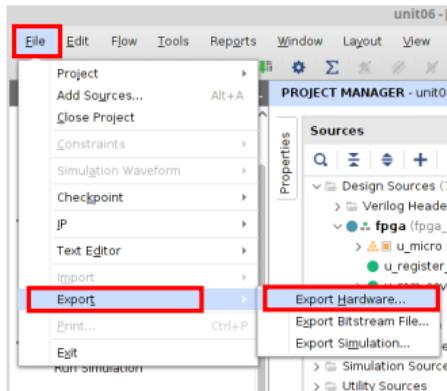


*Configuramos cuantos cores queremos utilizar para generar el binario.*

## Crear la aplicación en Vitis

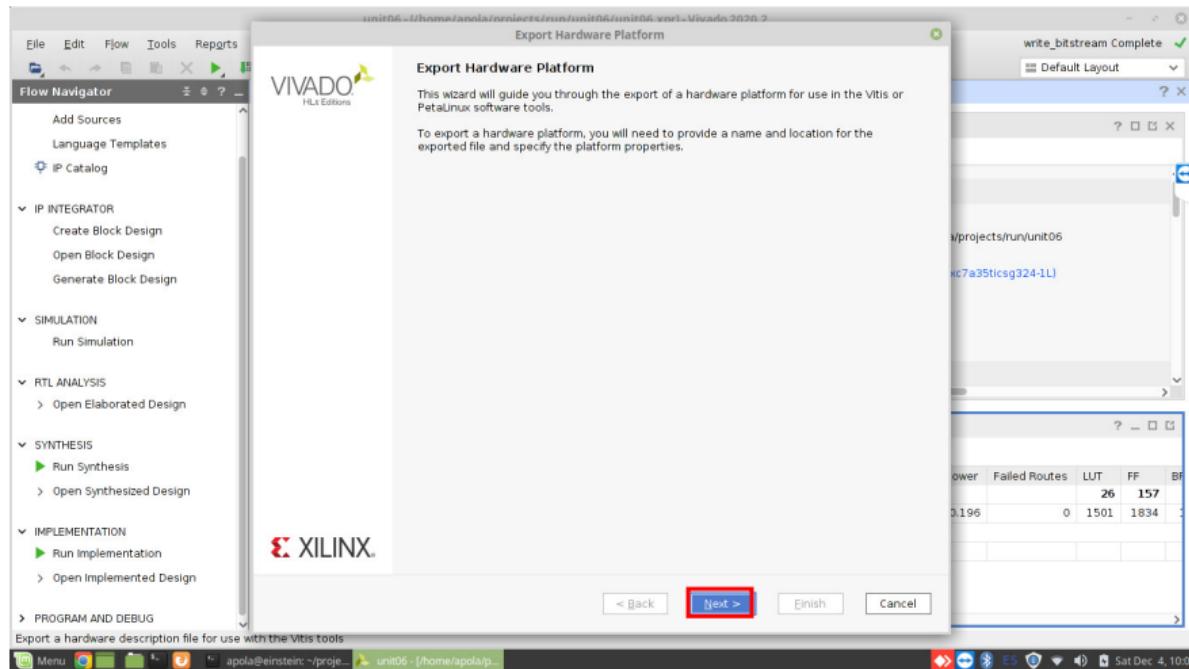


# Crear la aplicación en Vitis



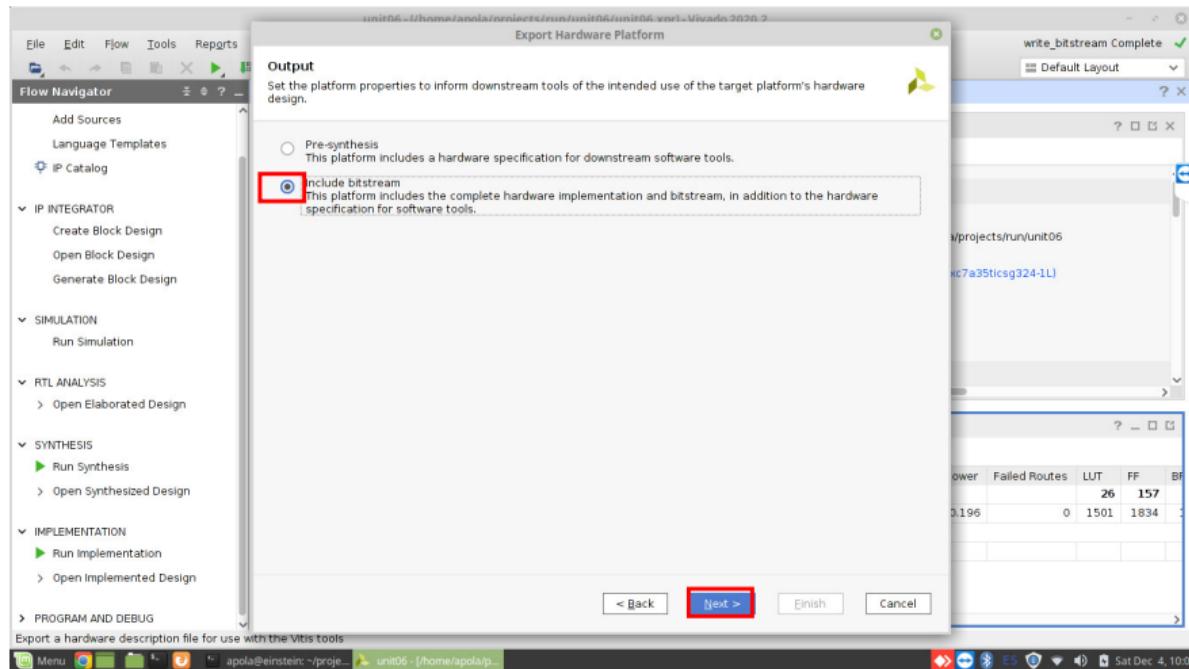
*Exportar el hardware desde la opción “File” de la barra de herramientas.*

# Crear la aplicación en Vitis



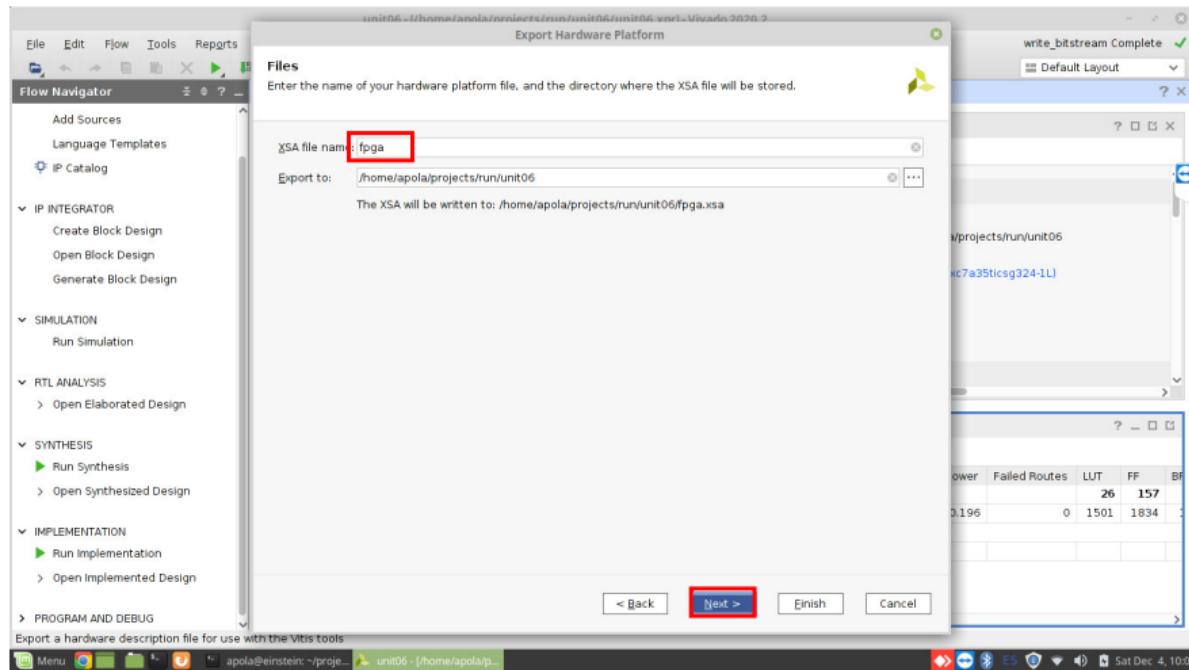
Pulsar NEXT.

# Crear la aplicación en Vitis



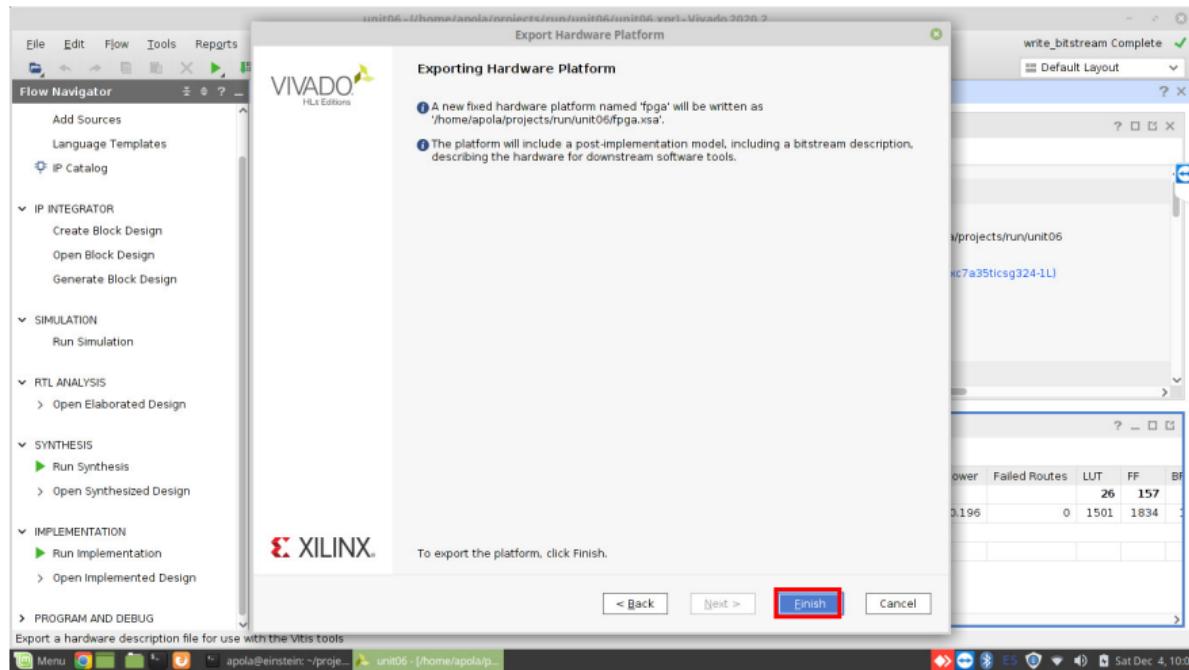
Incluir el bitstream.

# Crear la aplicación en Vitis



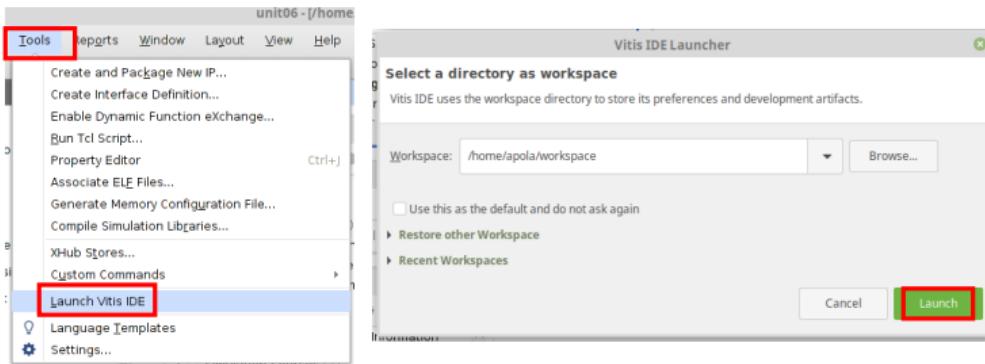
Definir el nombre del XSA (dejarlo por defecto).

# Crear la aplicación en Vitis



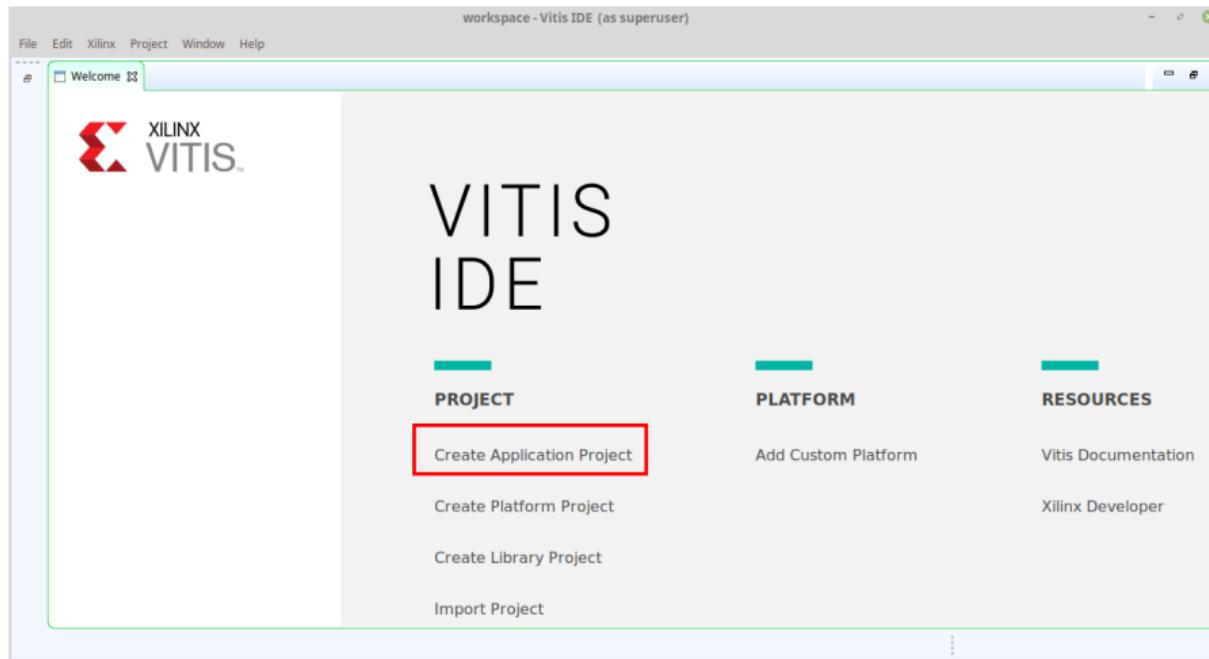
Pulsar FINISH.

# Crear la aplicación en Vitis



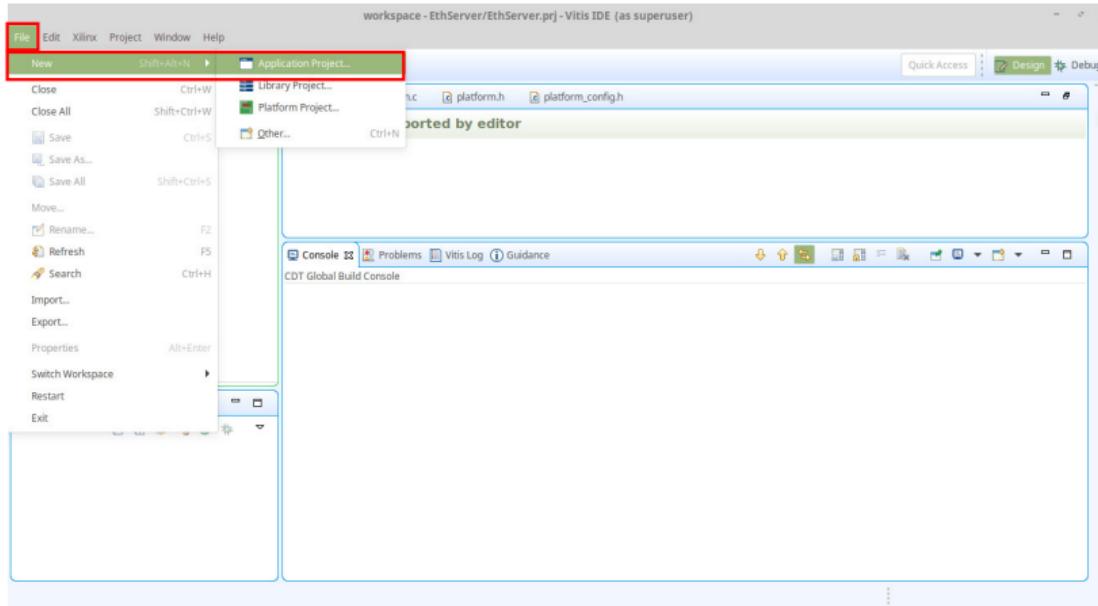
Ejecutar Vitis desde Vivado, seleccionando la opción Launch Vitis IDE.

# Crear la aplicación en Vitis



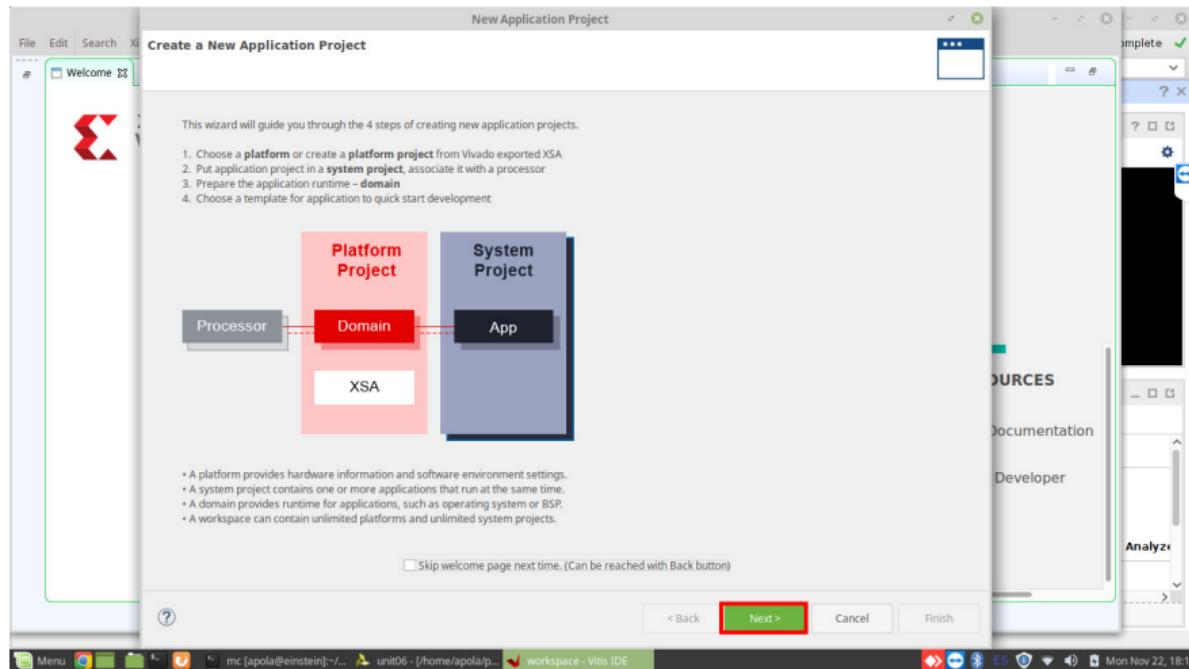
Entorno de trabajo en Vitis. Opción de inicio I: En la pagina de inicio seleccionar Create Application Project.

# Crear la aplicación en Vitis



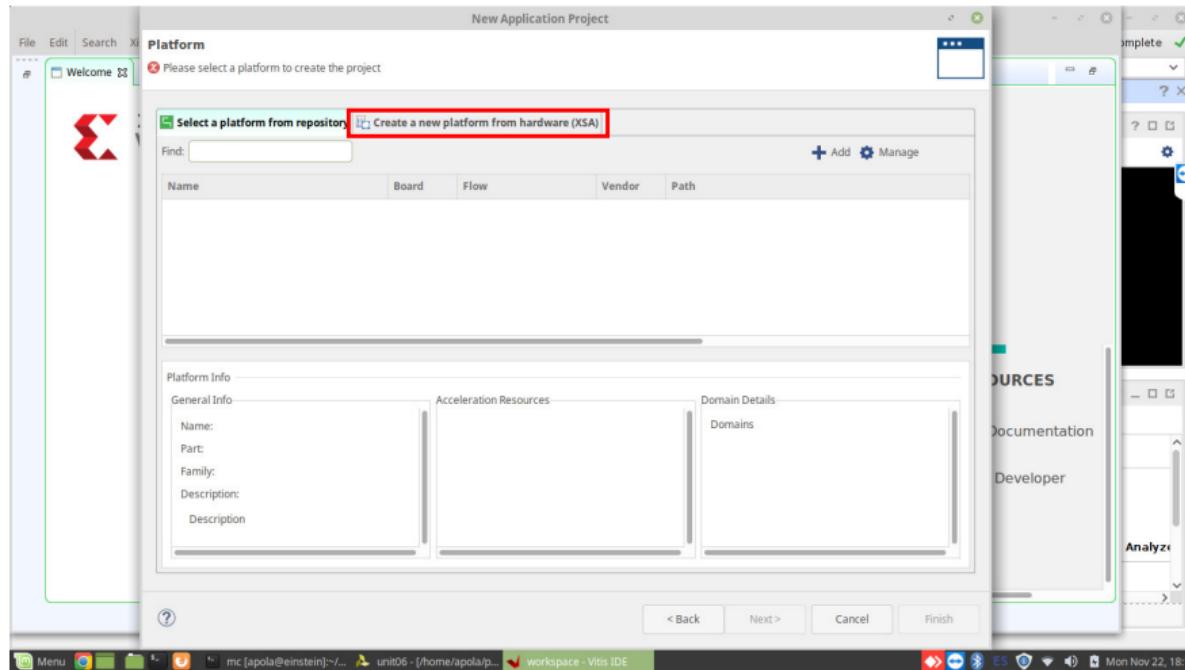
Entorno de trabajo en Vitis. Opción de inicio II: También puede iniciarse la nueva aplicación desde File-New-Application Project.

# Crear la aplicación en Vitis



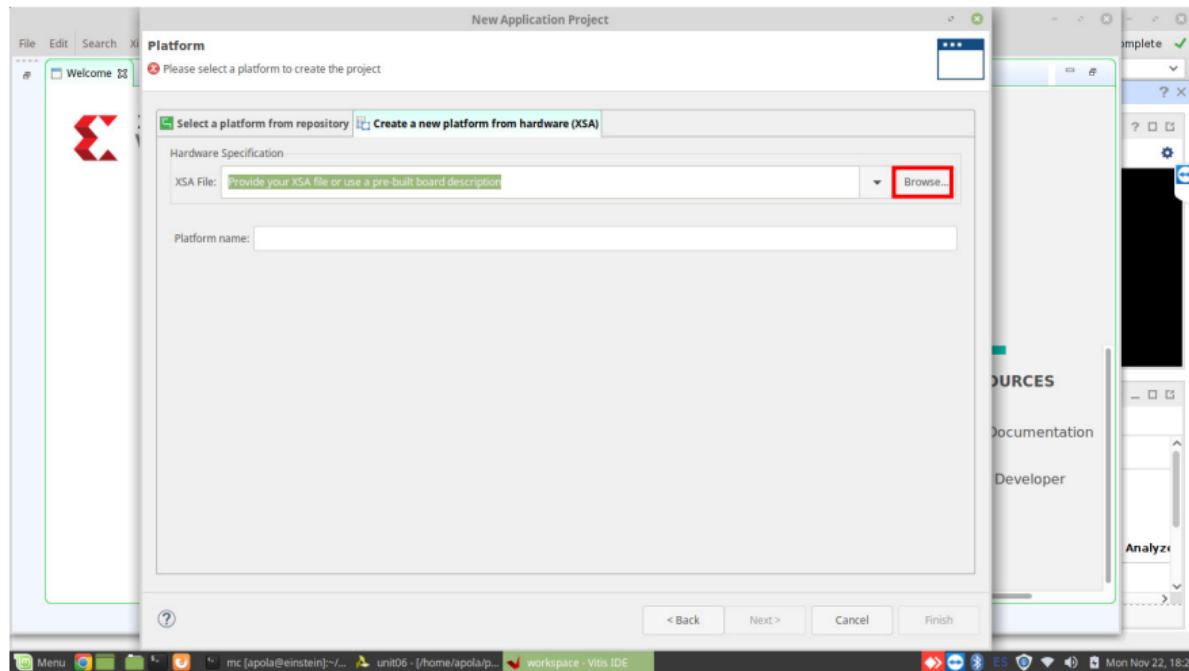
*Crear un nuevo proyecto.*

# Crear la aplicación en Vitis



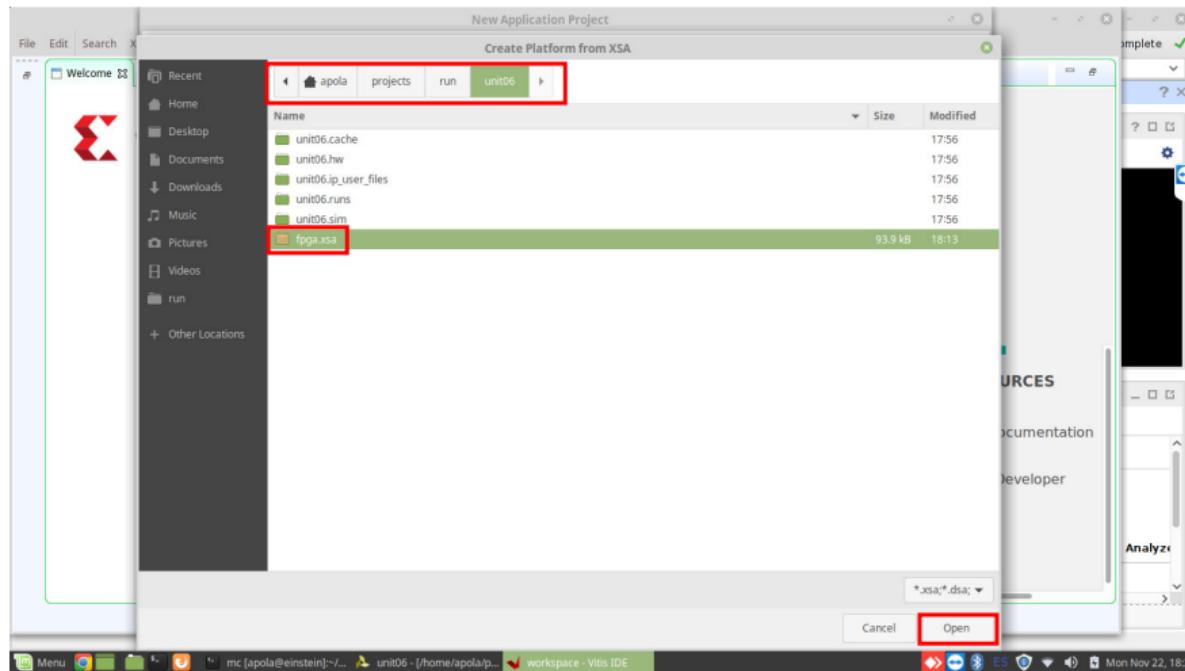
Crear una nueva plataforma importando el archivo XSA generado con el Vivado.

# Crear la aplicación en Vitis



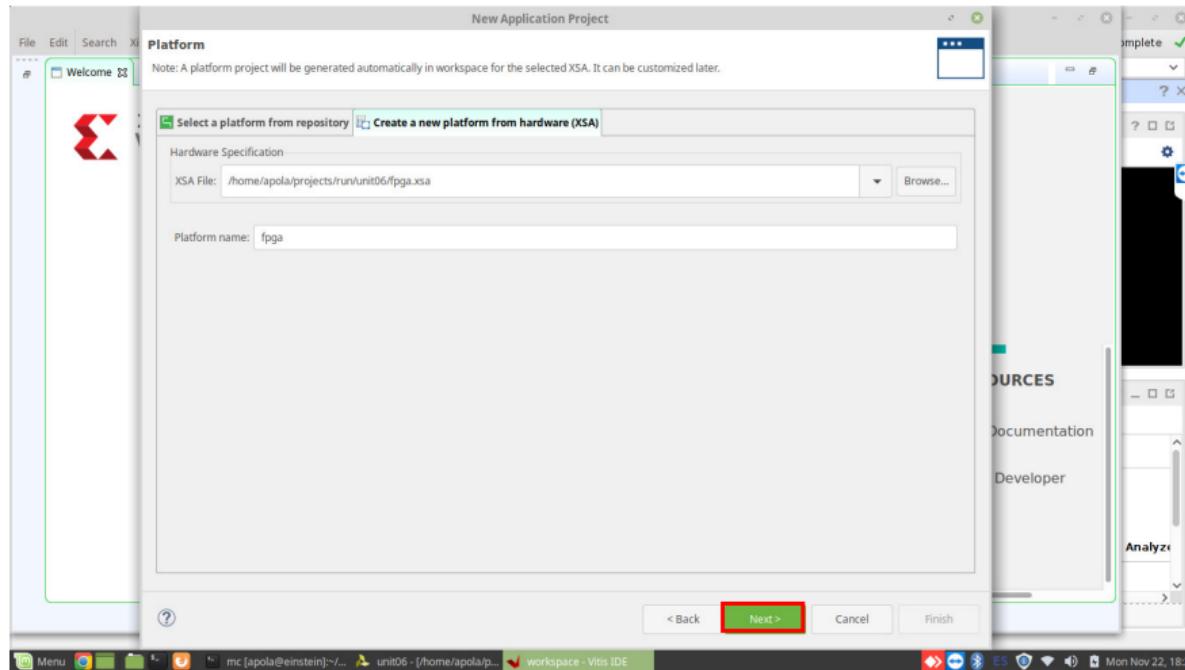
*Crear una nueva plataforma importando el archivo XSA generado con el Vivado.*

# Crear la aplicación en Vitis



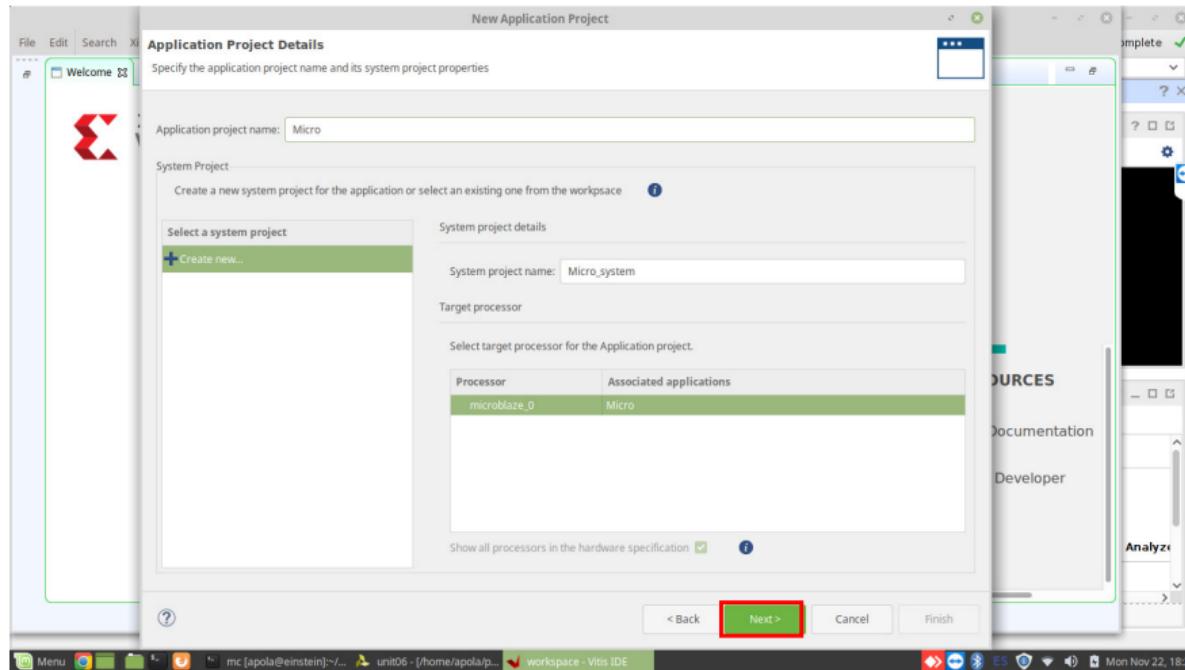
*Crear una nueva plataforma importando el archivo XSA generado con el Vivado.*

# Crear la aplicación en Vitis



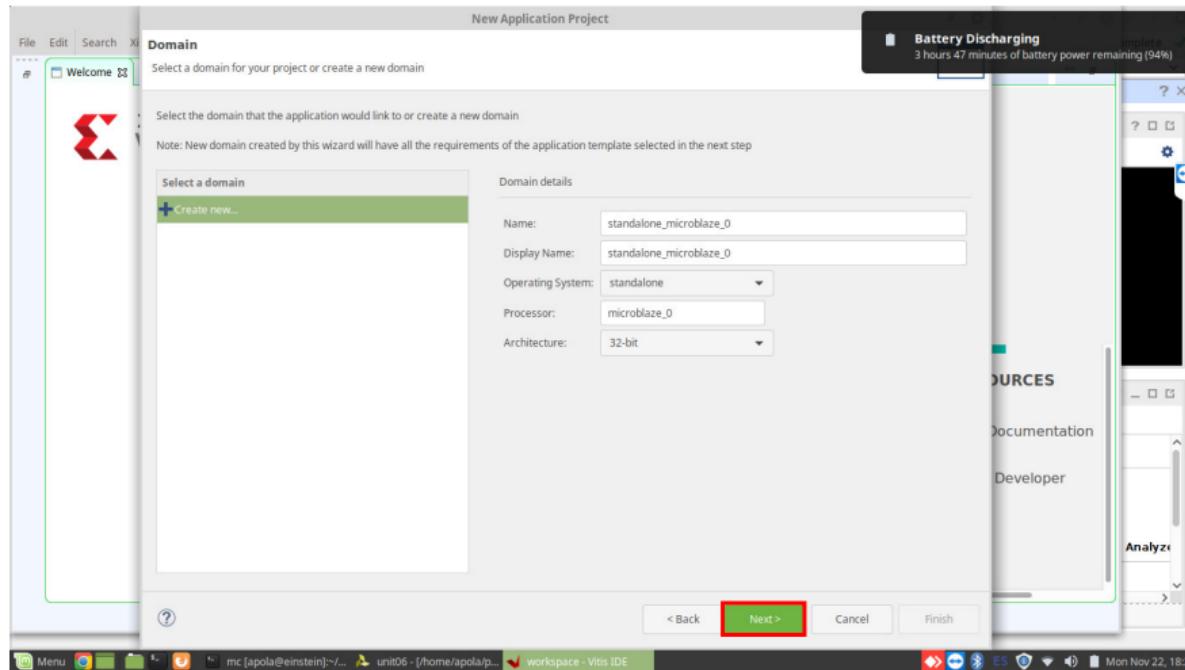
*Crear una nueva plataforma importando el archivo XSA generado con el Vivado.*

# Crear la aplicación en Vitis



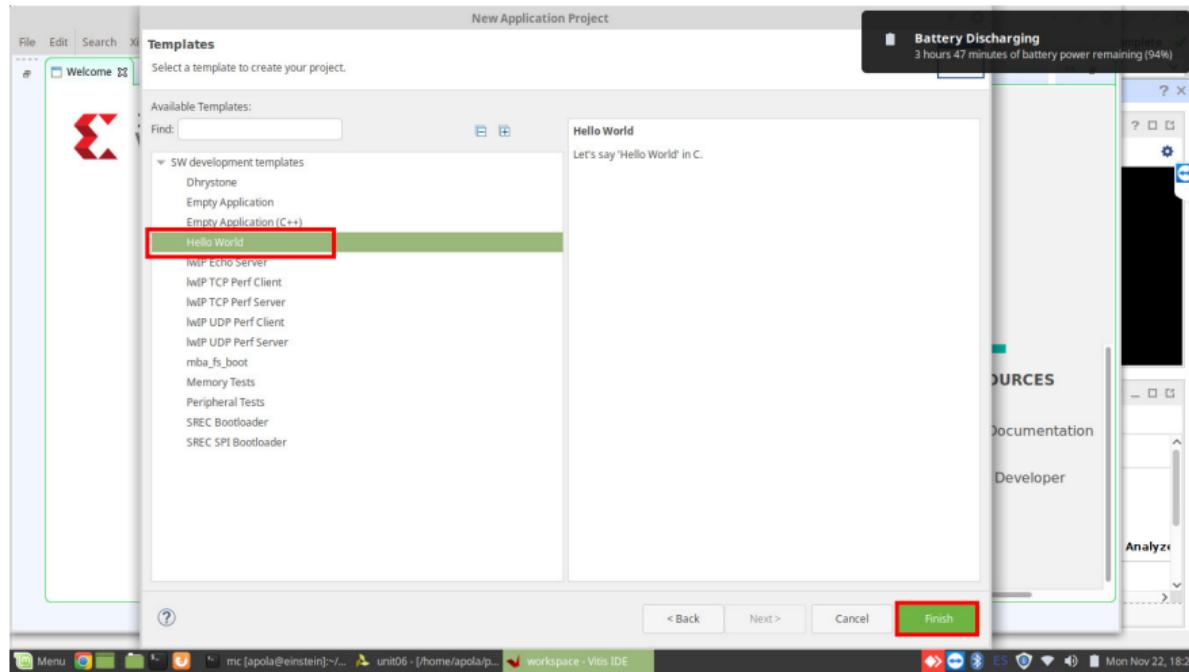
Asignar el nombre de la Aplicación (Ej. Micro).

# Crear la aplicación en Vitis



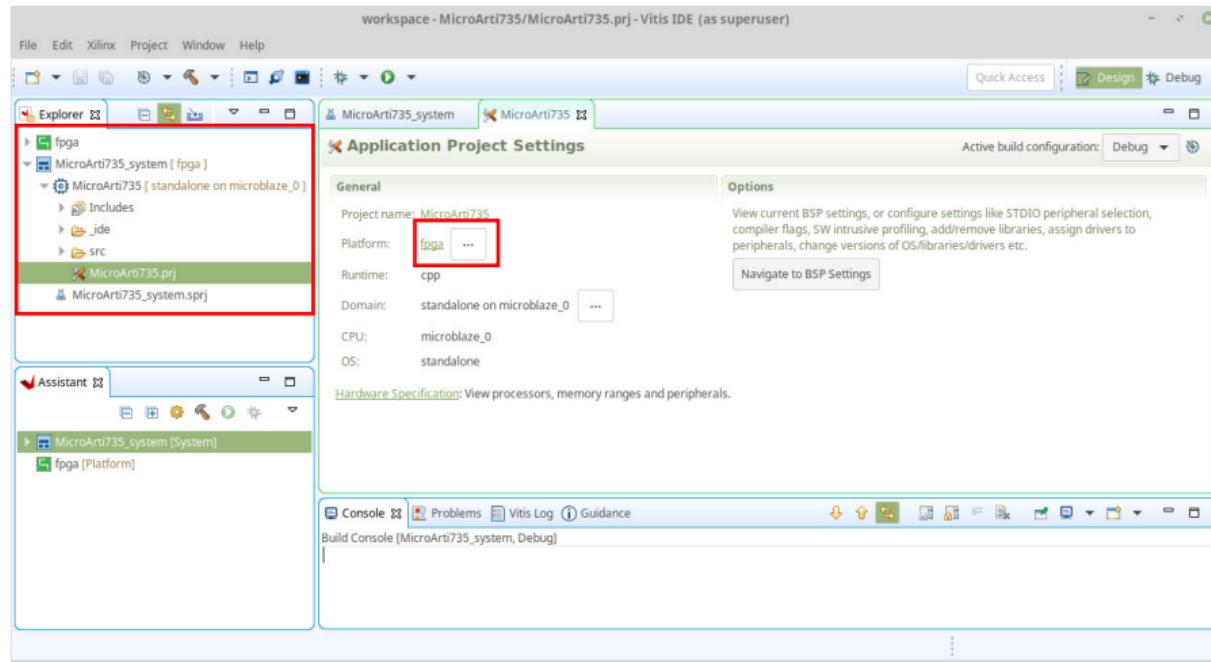
*Verificar las características del Micro.*

# Crear la aplicación en Vitis



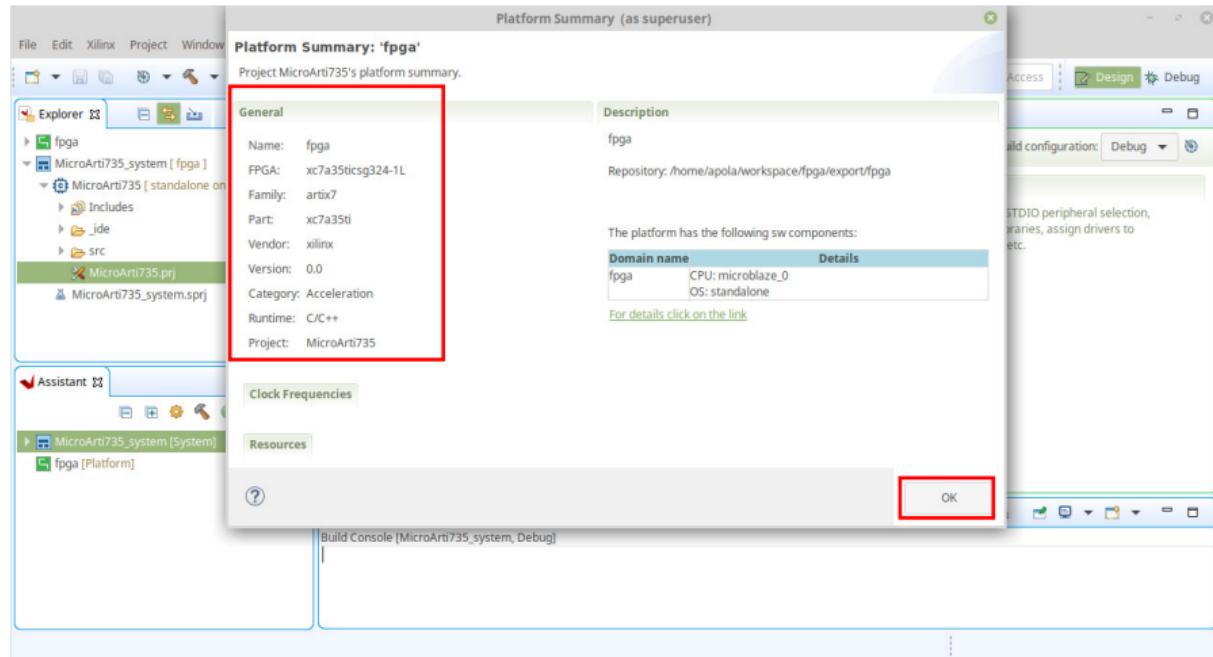
Asignar el proyecto Hello Word.

# Crear la aplicación en Vitis



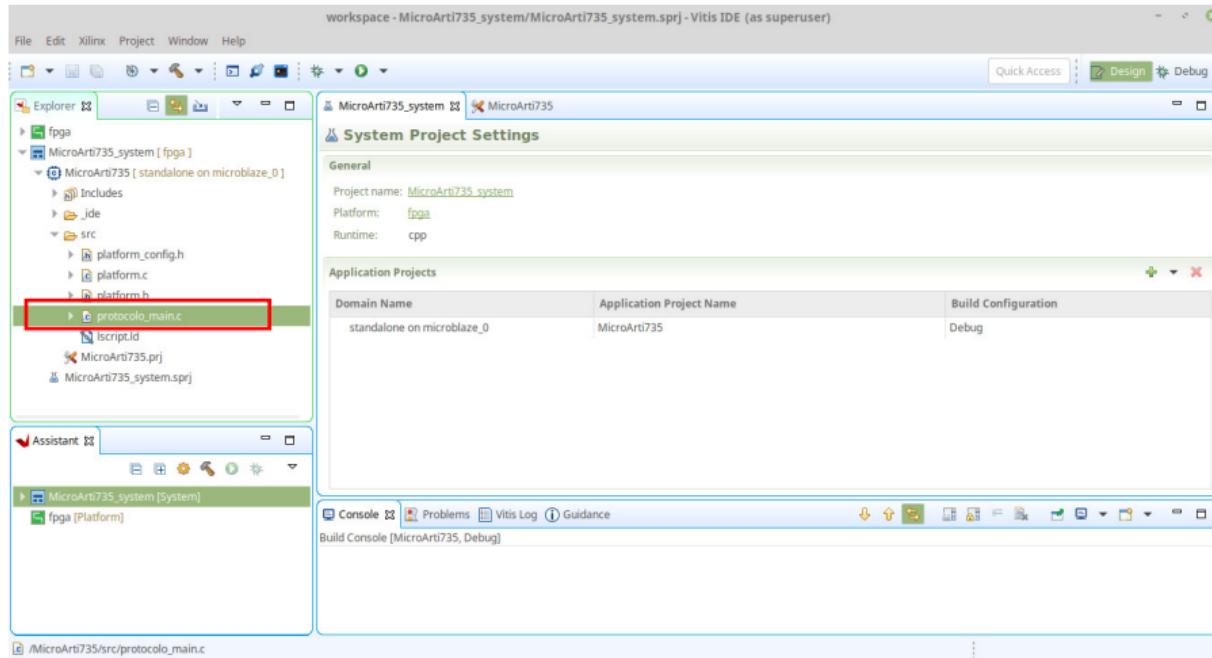
Se generan dos carpetas con el detalle del hardware y la aplicación en C seleccionada. Se puede verificar las características técnicas del diseño haciendo click en el nombre del hardware.

# Crear la aplicación en Vitis



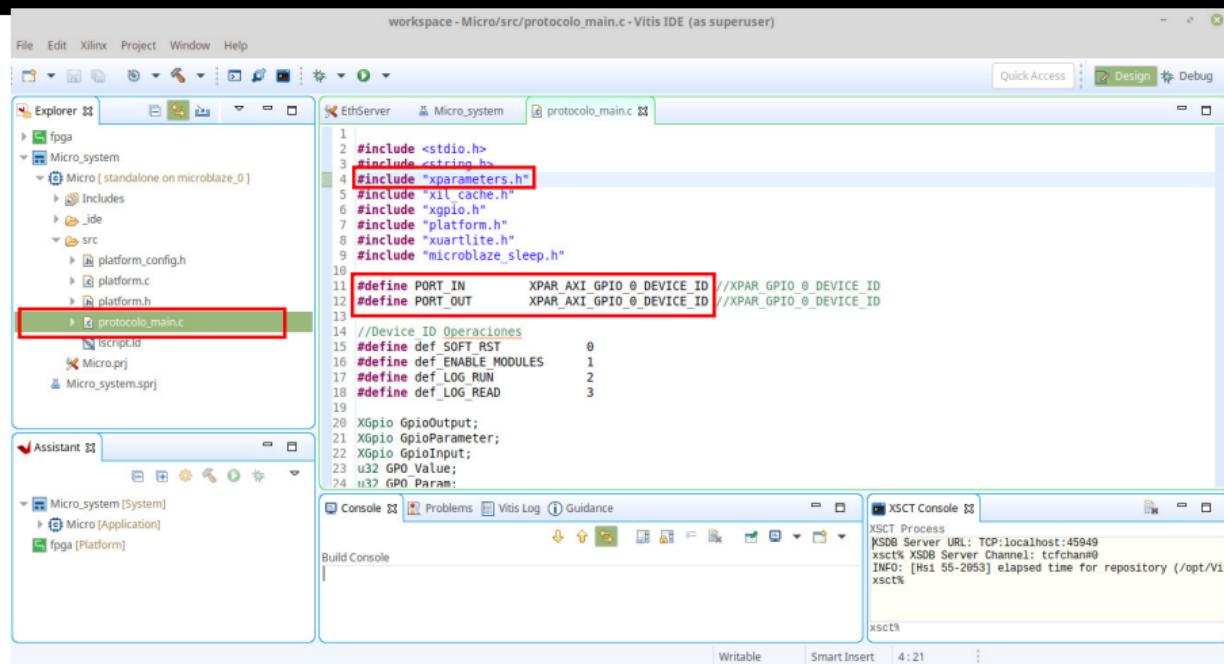
*Detalles de la aplicación.*

# Crear la aplicación en Vitis



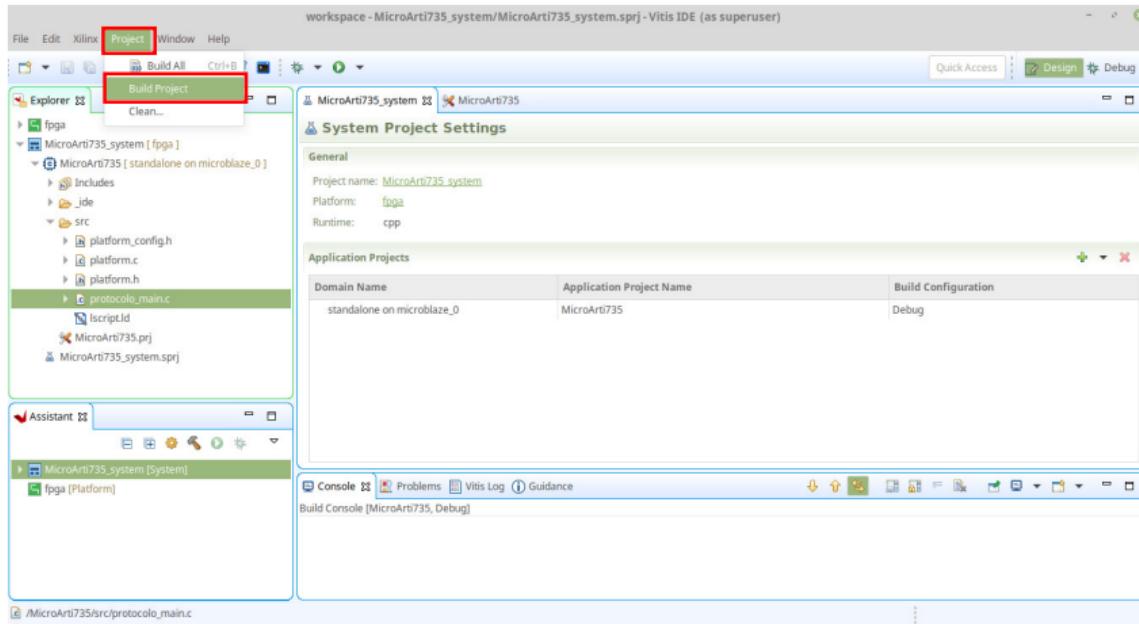
Reemplazar el archivo "Hello Word" con "protocolo\_main" desde el dirección del proyecto. Ej.  
"/home/apola/workspace/MicroArti735/src/"

# Crear la aplicación en Vitis



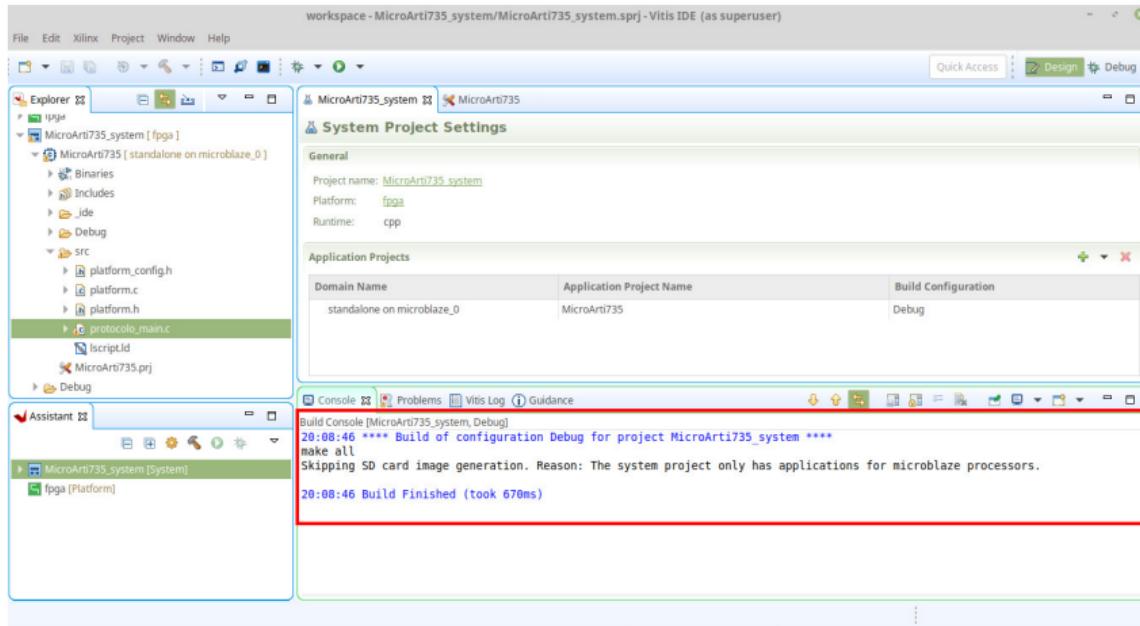
Verificar la definición de los puertos GPIO desde el archivo “`xparameters.h`”. En caso de error en la compilación, debemos buscar en el archivo “`xparameters.h`” la definición que contenga las palabras claves `XPAR`, `AXI`, `GPIO`, `DEVICE` e `ID`.

# 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.

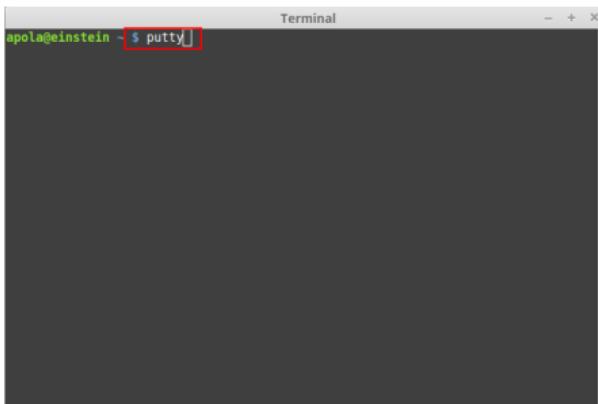


## Tunel, Programar la FPGA y Ejecutar la Aplicación en Forma Remota

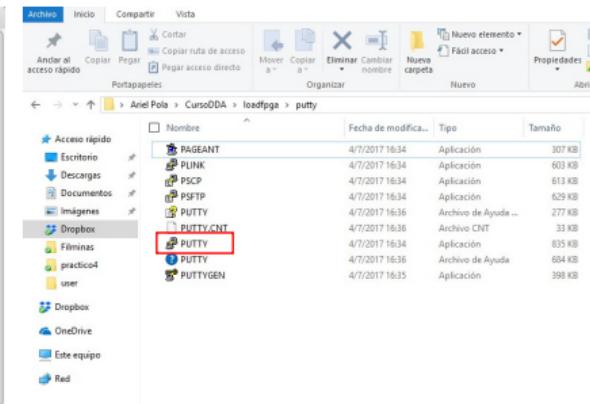
# Tunel y Asignación de FPGA

## 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 ícono de Putty.

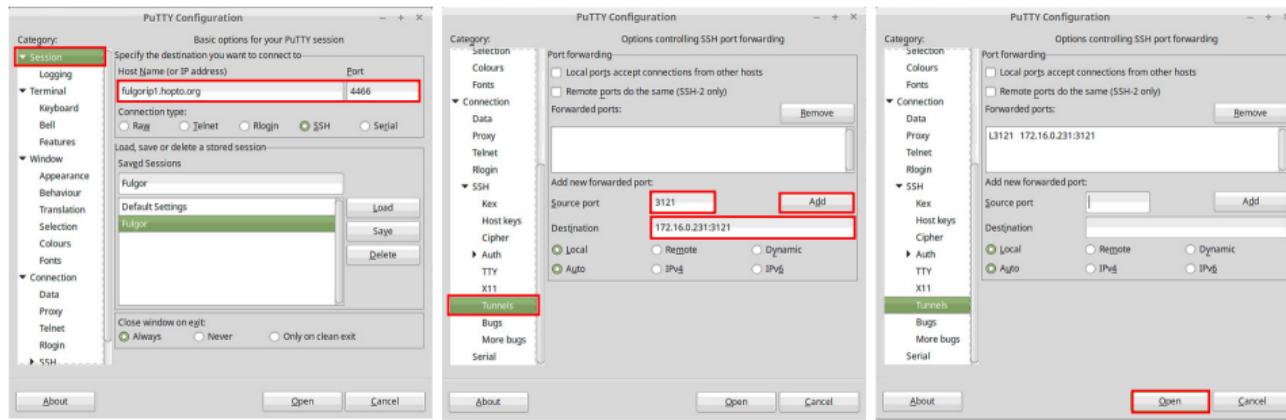


Linux



Windows

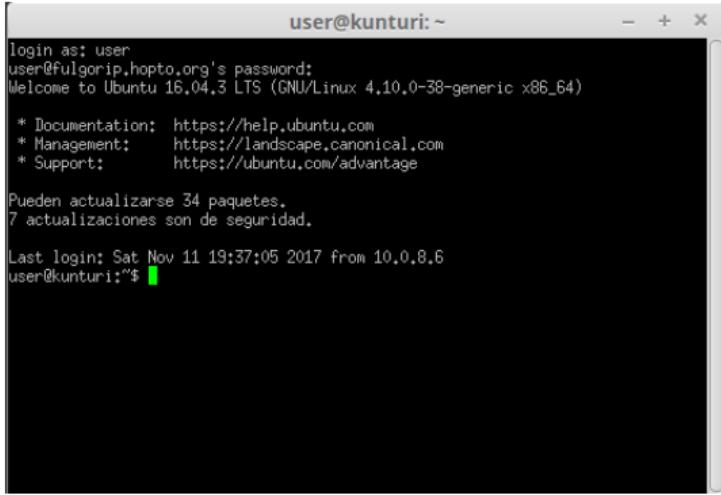
# Tunel y Asignación de FPGA



- 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 tunel.

# Tunel 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.

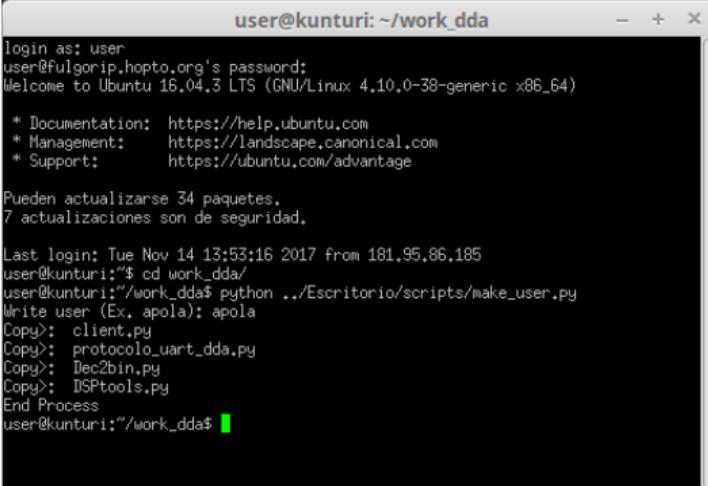
Last login: Sat Nov 11 19:37:05 2017 from 10.0.8.6
user@kunturi:~$
```

# Tunel 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$
```

# Tunel 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 estan 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.

# Tunel y Asignación de FPGA

## 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>/
```

# Tunel y Asignación de FPGA

The image displays two side-by-side terminal windows. Both windows have a black background and white text. They show the same sequence of commands being run:

```
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: USB3-210319A2CE0DA
-----
Write Text to check Session or Exit to close: [redacted]
```

The right terminal window shows the continuation of the session:

```
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.

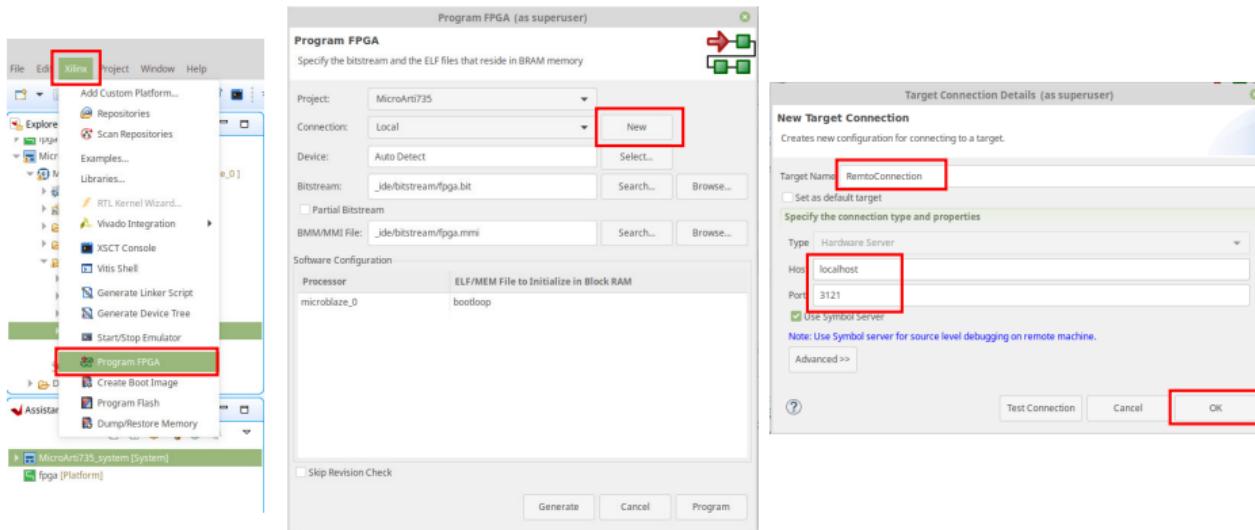
Last login: Tue Nov 14 14:23:29 2017 from 181.95.86.185
user@kunturi:~$ cd work_dda/apola/scripts/
user@kunturi:~/work_dda/apola/scripts$ python client.py

Write-> Exit <- to close the session

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

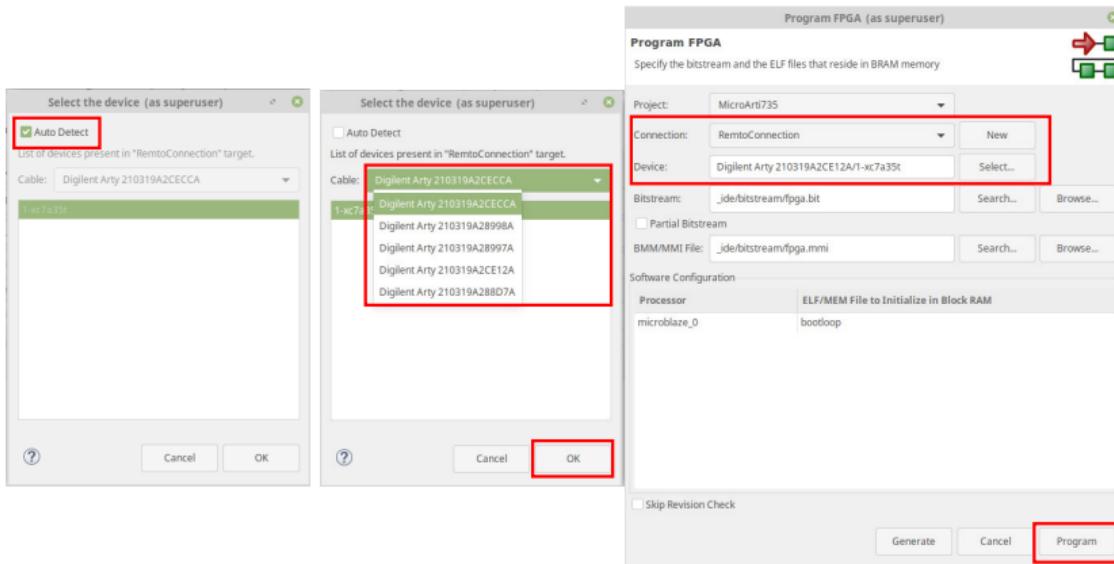
Las figuras muestran la asignación correcta de puerto (izq.) y los puertos ocupados (der.).

# Programar la FPGA



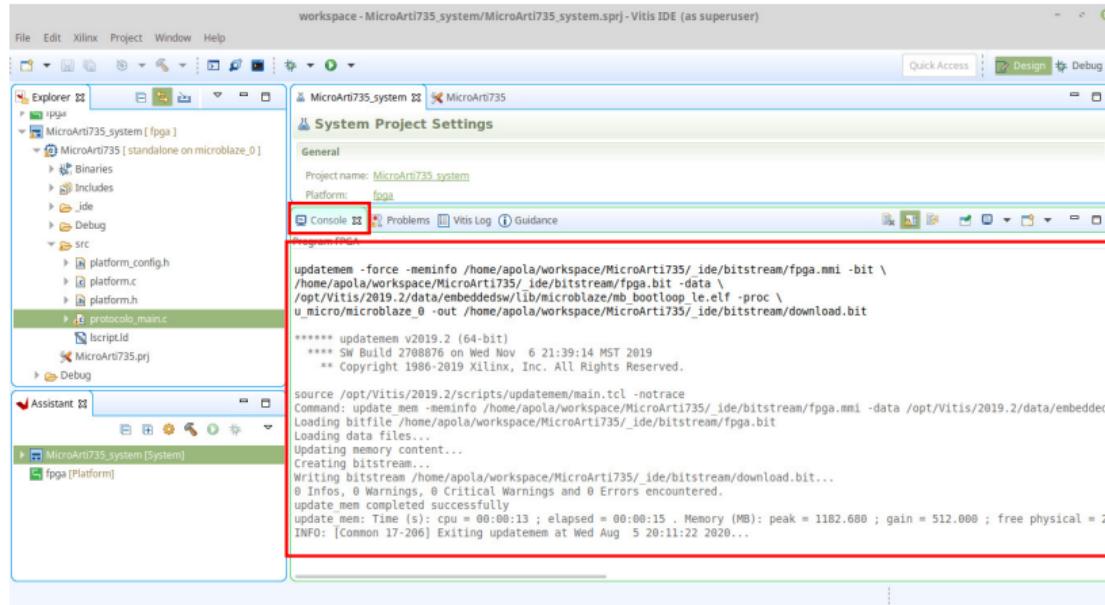
Programar la FPGA. Crear un nuevo Server, dar un nombre y colocar la dirección IP del server remoto. Para nuestro caso, el cual realiza un forward del puerto 3121 se debe colocar LOCALHOST como dirección de host.

# Programar la FPGA



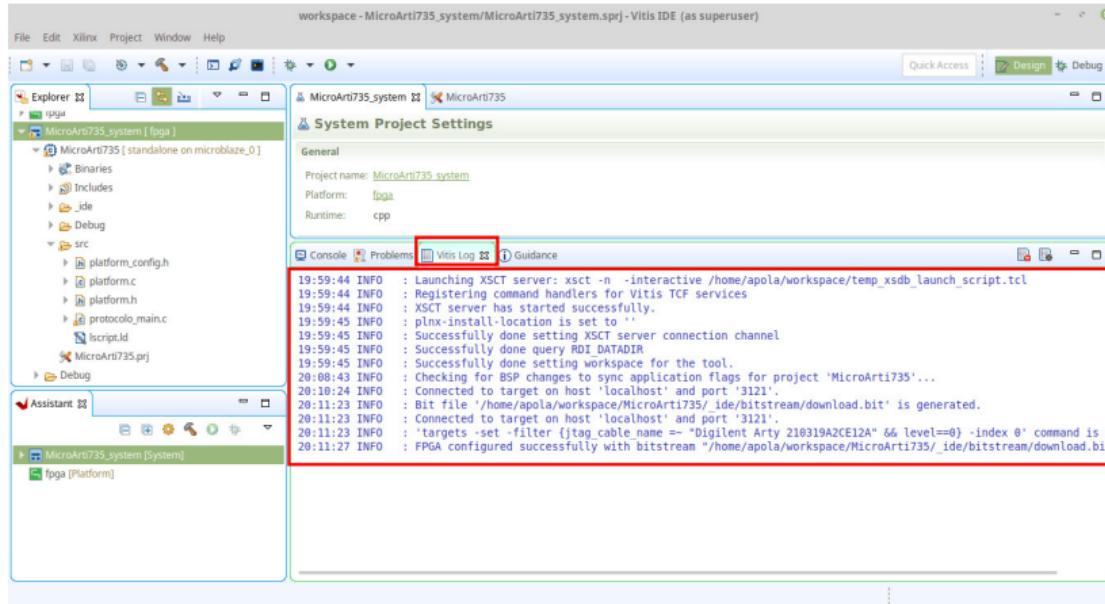
Desactivar AutoDetect, seleccionar el ID de la FPGA y verificar la información.

# Programar la FPGA



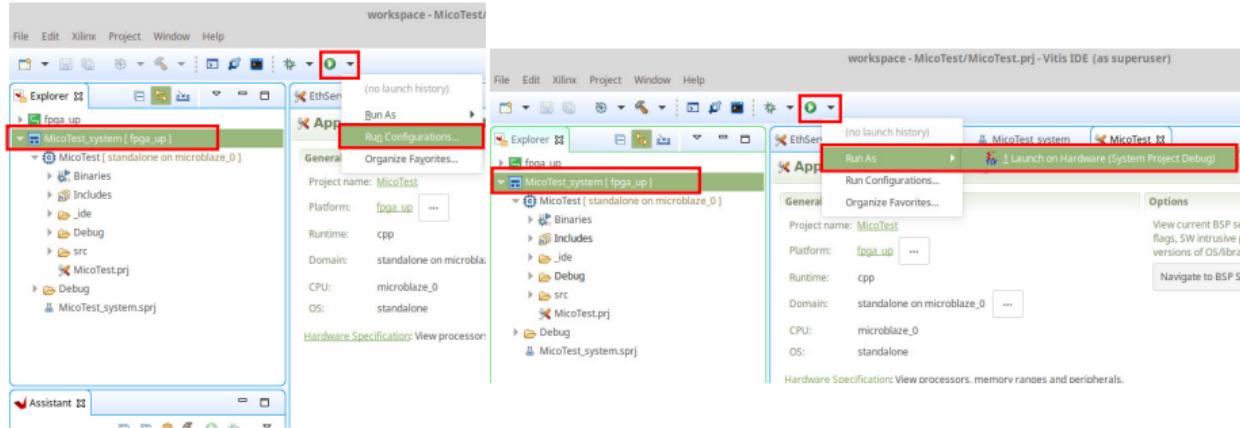
En la “consola” se observará la generación de un nuevo binario.

# Programar la FPGA



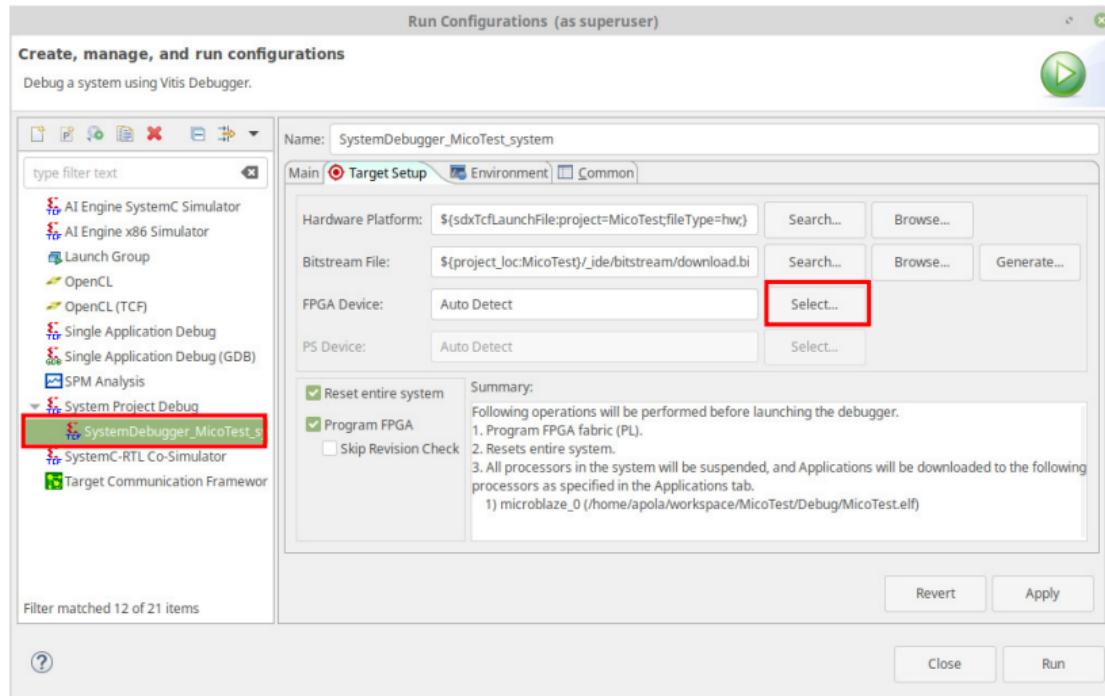
En “Vitis Log” se detallará la programación.

# Ejecutar la Aplicación



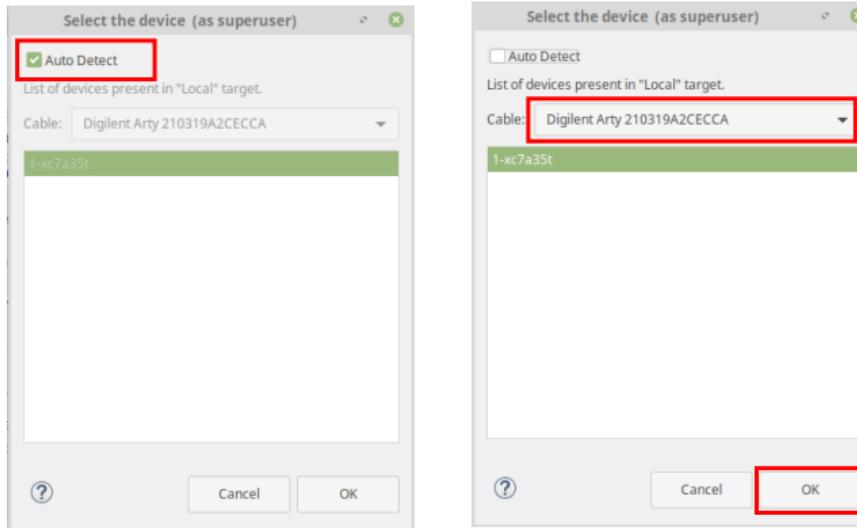
Antes de ejecutar la aplicación es necesario setear el dispositivo que se quiere correr. Para ello necesitamos acceder a Run Configuration. **Nota: En caso que no aparezcan las opciones que se detallan en la figura de la siguiente pagina, debemos ejecutar al menos una vez la aplicación. En este caso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

# Ejecutar la Aplicación



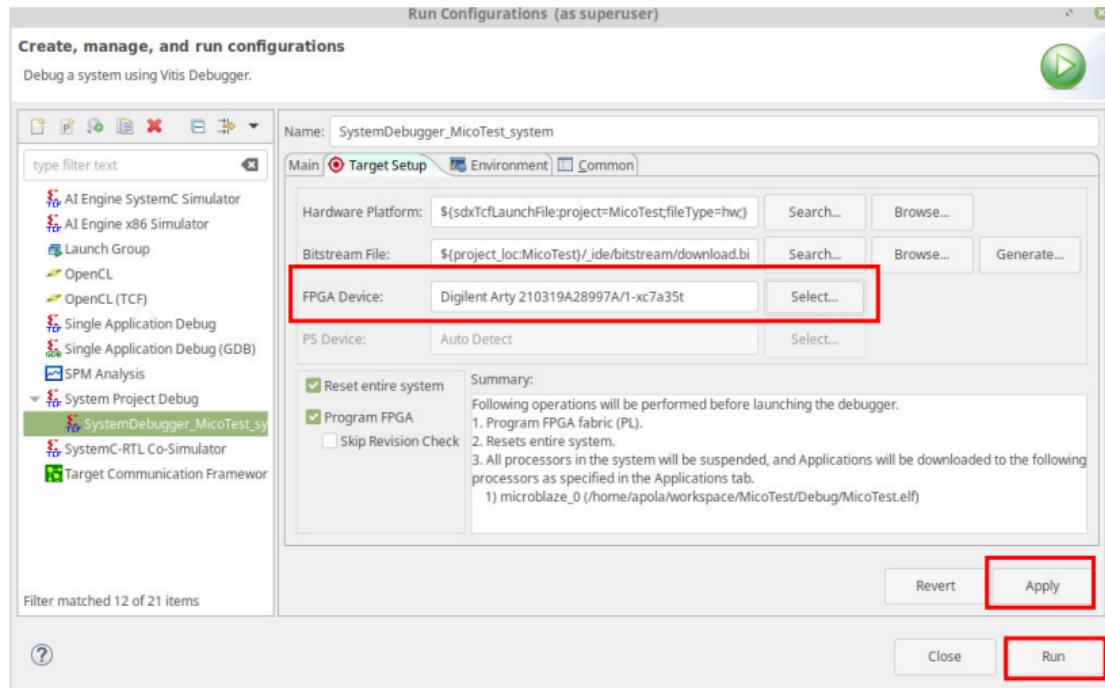
*En la opción SystemDebugger, cambiar el dispositivo (FPGA Device).*

# Ejecutar la Aplicación



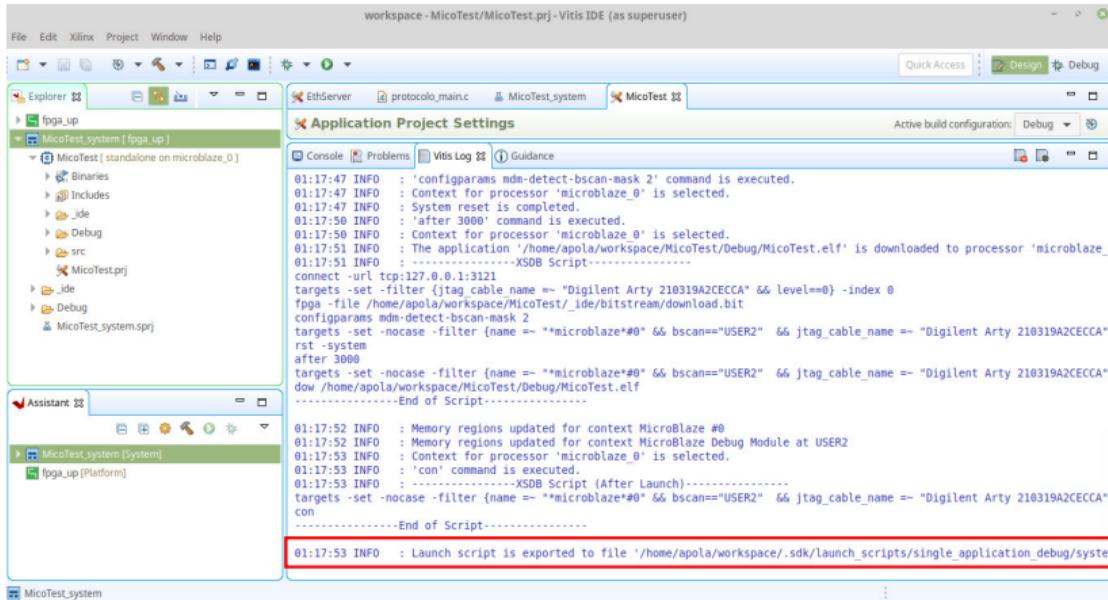
Seleccionar una FPGA.

# Ejecutar la Aplicación



Verificar el ID, aplicar los cambios y ejecutar.

# Ejecutar la Aplicación



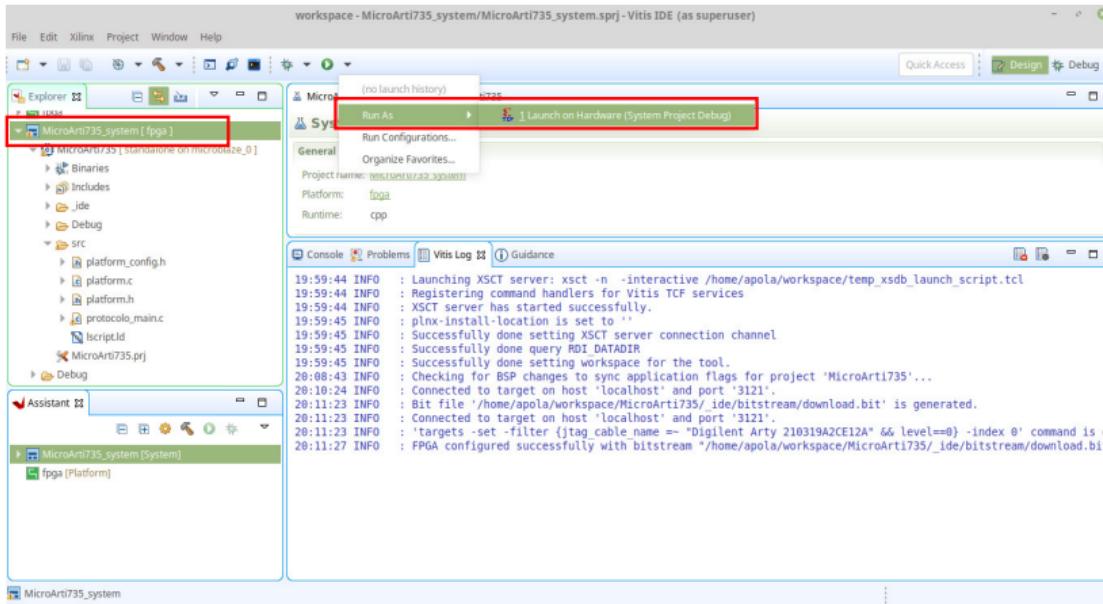
The screenshot shows the Vitis IDE interface with the following details:

- File menu:** File, Edit, Xilinx, Project, Window, Help.
- Toolbars:** Quick Access, Design, Debug.
- Explorer View:** Shows the project structure:
  - fpga\_up
  - MicoTest\_system [ fpga\_up ]
    - MicoTest [ standalone on microblaze\_0 ]
      - Binaries
      - Includes
      - IDE
      - Debug
      - src
        - MicoTest.prj
    - IDE
    - Debug
    - MicoTest\_system.sprj
  - MicoTest\_system [System]
  - fpga\_up [Platform]
  - MicoTest\_system
- Application Project Settings Tab:** Shows the active build configuration as Debug.
- Console Tab:** Displays the following log output:

```
01:17:47 INFO  : 'configparam mdm-detect-bscan-mask 2' command is executed.
01:17:47 INFO  : Context for processor 'microblaze_0' is selected.
01:17:47 INFO  : System reset is completed.
01:17:50 INFO  : 'after 3000' command is executed.
01:17:50 INFO  : Context for processor 'microblaze_0' is selected.
01:17:51 INFO  : The application '/home/apola/workspace/MicoTest/Debug/MicoTest.elf' is downloaded to processor 'microblaze_0'.
01:17:51 INFO  : -----XSDB Script-----
connect -url tcp:127.0.0.1:3121
targets -set -filter {jtag cable name == "Digilent Arty 210319A2CECCA" && level==0} -index 0
fpga -file /home/apola/workspace/MicoTest/ide/bitstream/download.bit
configparam mdm-detect-bscan-mask 2
targets -set -nocase -filter {name == "*microblaze*#0" && bscan=="USER2" && jtag_cable_name == "Digilent Arty 210319A2CECCA"}
rst -system
after 3000
targets -set -nocase -filter {name == "*microblaze*#0" && bscan=="USER2" && jtag_cable_name == "Digilent Arty 210319A2CECCA"}
dow /home/apola/workspace/MicoTest/Debug/MicoTest.elf
-----End of Script-----
01:17:52 INFO  : Memory regions updated for context MicroBlaze #0
01:17:52 INFO  : Memory regions updated for context MicroBlaze Debug Module at USER2
01:17:53 INFO  : Context for processor 'microblaze_0' is selected.
01:17:53 INFO  : 'con' command is executed.
01:17:53 INFO  : -----XSDB Script (After Launch)-----
targets -set -nocase -filter {name == "*microblaze*#0" && bscan=="USER2" && jtag_cable_name == "Digilent Arty 210319A2CECCA"}
con
-----End of Script-----
01:17:53 INFO  : Launch script is exported to file '/home/apola/workspace/.sdk/launch_scripts/single_application_debug/system.launch'
```

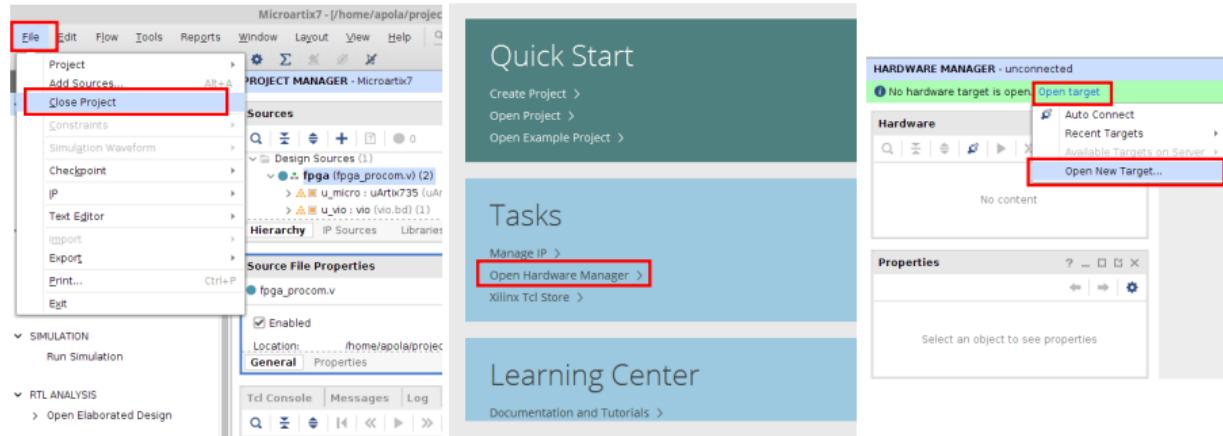
*El mensaje Launch verifica que se esta corriendo la aplicación en el microProcesador.*

# Ejecutar la Aplicación



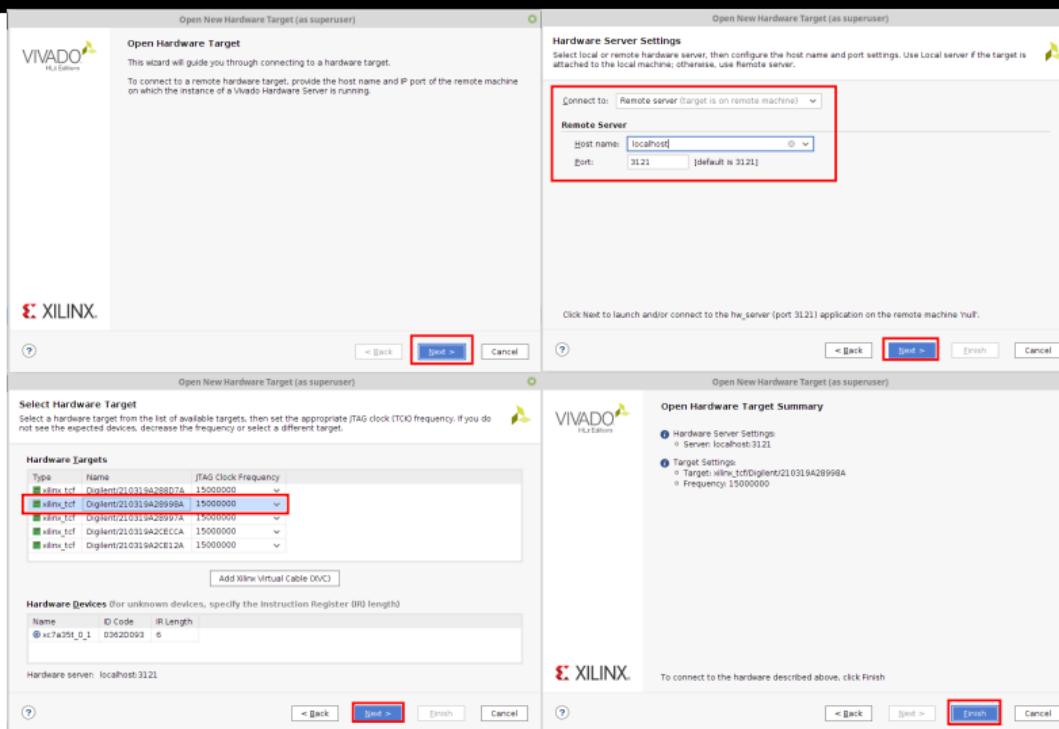
En caso de necesitar ejecutar varias veces la aplicación y una vez que se han ejecutado los pasos anteriores, basta con lanzar nuevamente la aplicación.

# Cargar parámetros del VIO



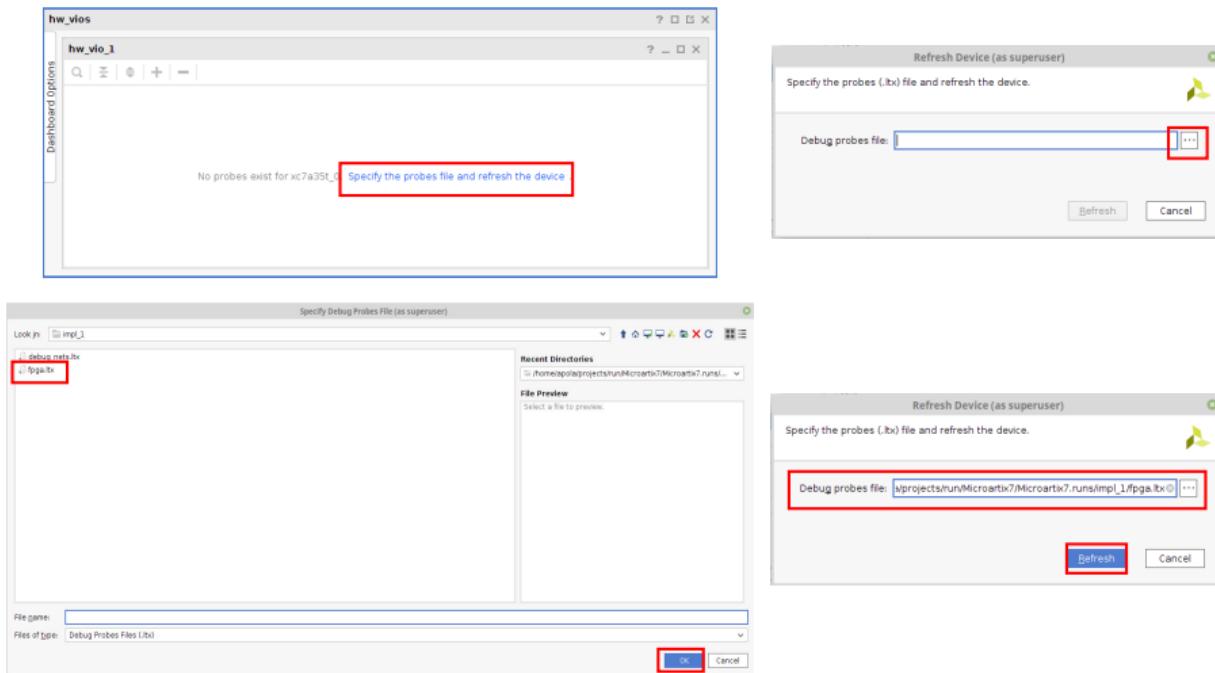
Volver a la ventana del Vivado, cerrar el proyecto, abrir el controlador de hardware y seleccionar un nuevo target.

# Cargar parámetros del VIO



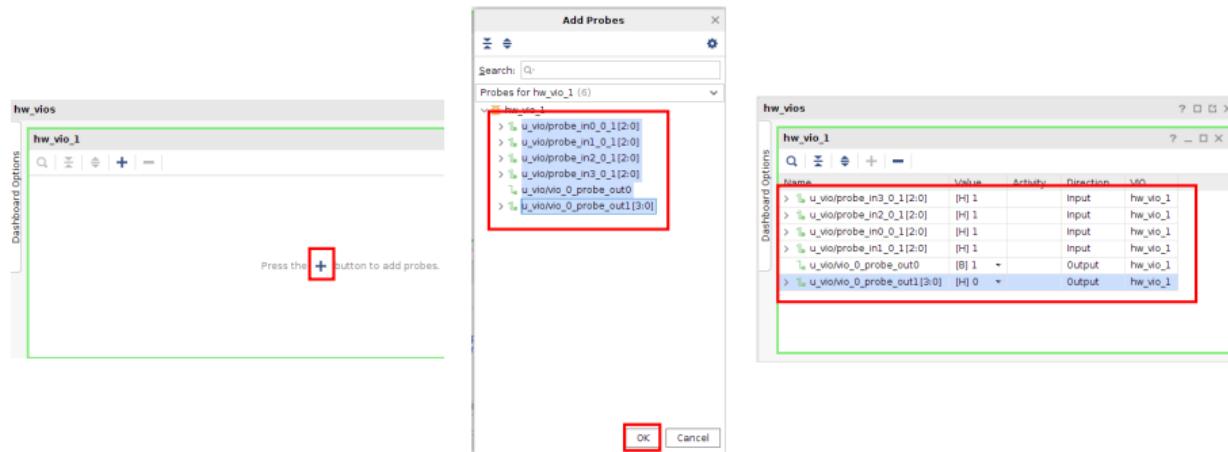
Crear un nuevo servidor cuya IP sea LOCALHOST y seleccionar la FPGA con el ID que previamente el servidor les entrega.

# Cargar parámetros del VIO



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

# Cargar parámetros del VIO



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

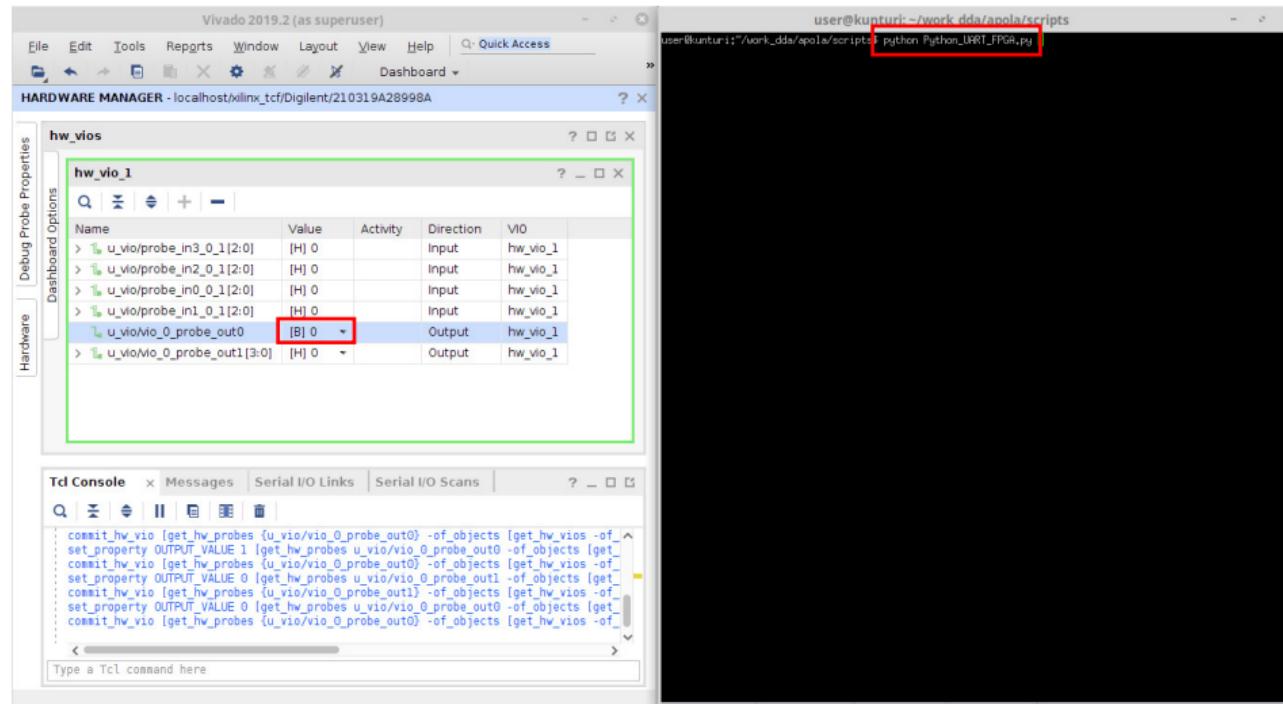
# Asignar el Puerto USB



A screenshot of a terminal window titled "mc [user@wanpu]:/home/apola/Escritorio/Python". The window contains a single line of text: "\$ python3 Python\_UART\_FPGA.py 5". The entire line is highlighted with a red rectangle.

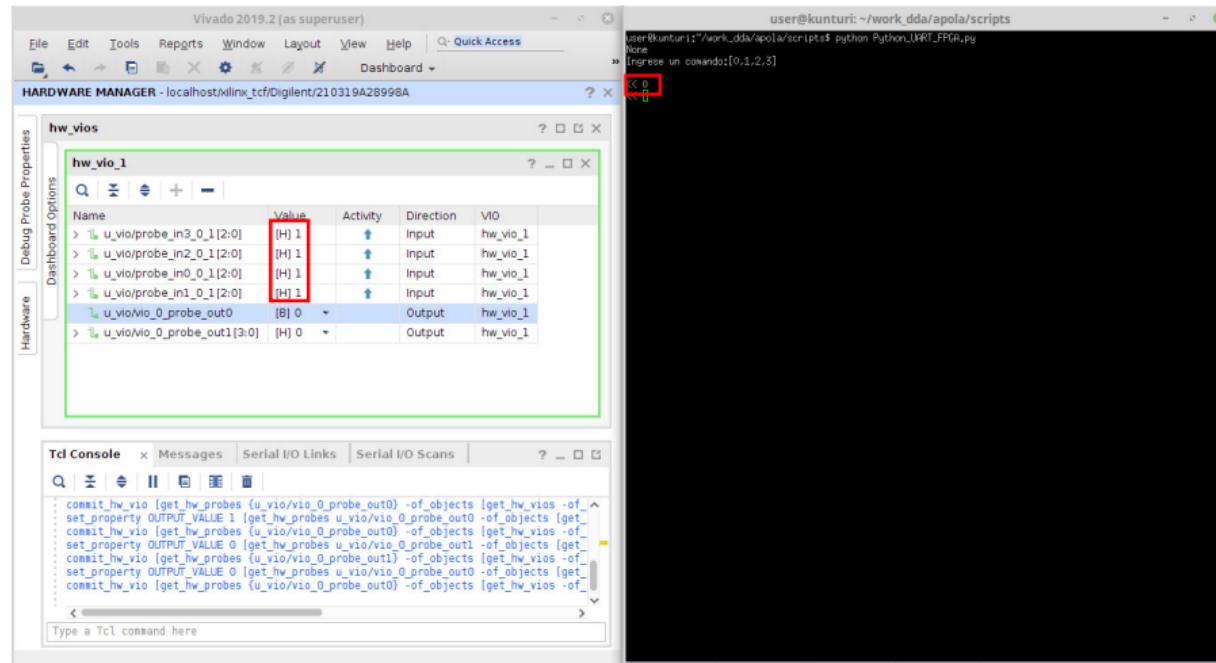
En un nuevo terminal de PUTTY, ejecutamos el script *Python\_UART\_FPGA.py* asignando el puerto *UART*. Ejecutar la aplicación con **python3**

# VIO y Python



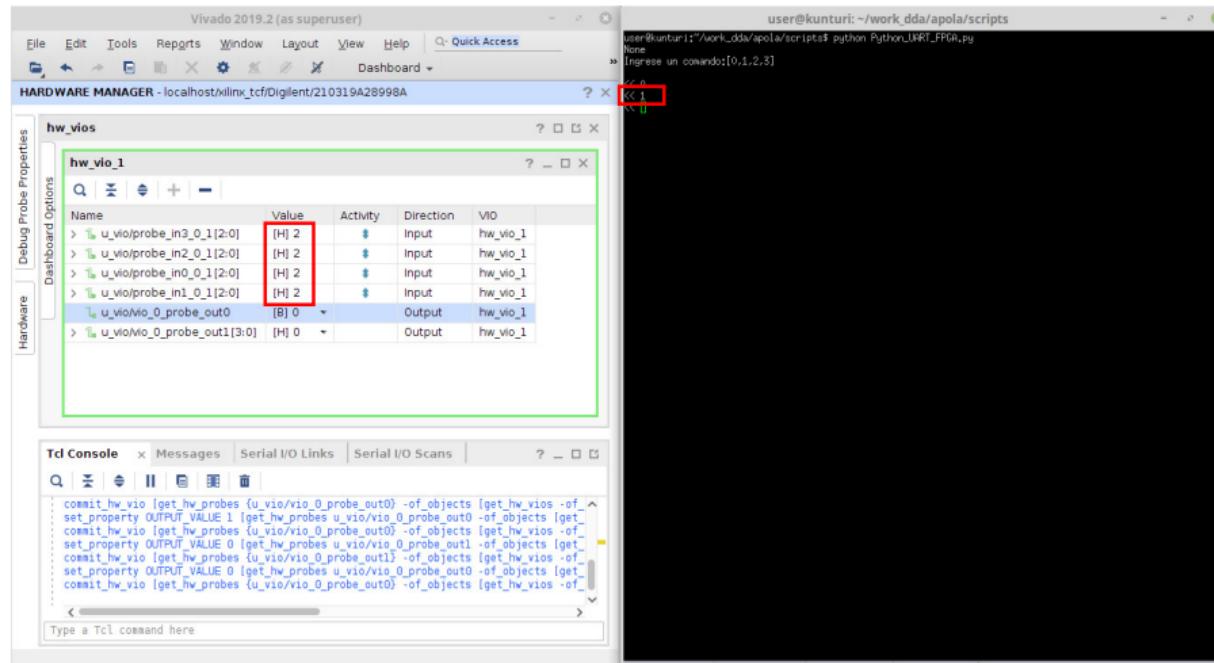
El puerto remarcado en el VIO es el que habilita si leemos el estado de los switch y lo que se carga desde el VIO (0: VIOCtrLEnb - 1: Switch). Ejecutamos el python.

# VIO y Python



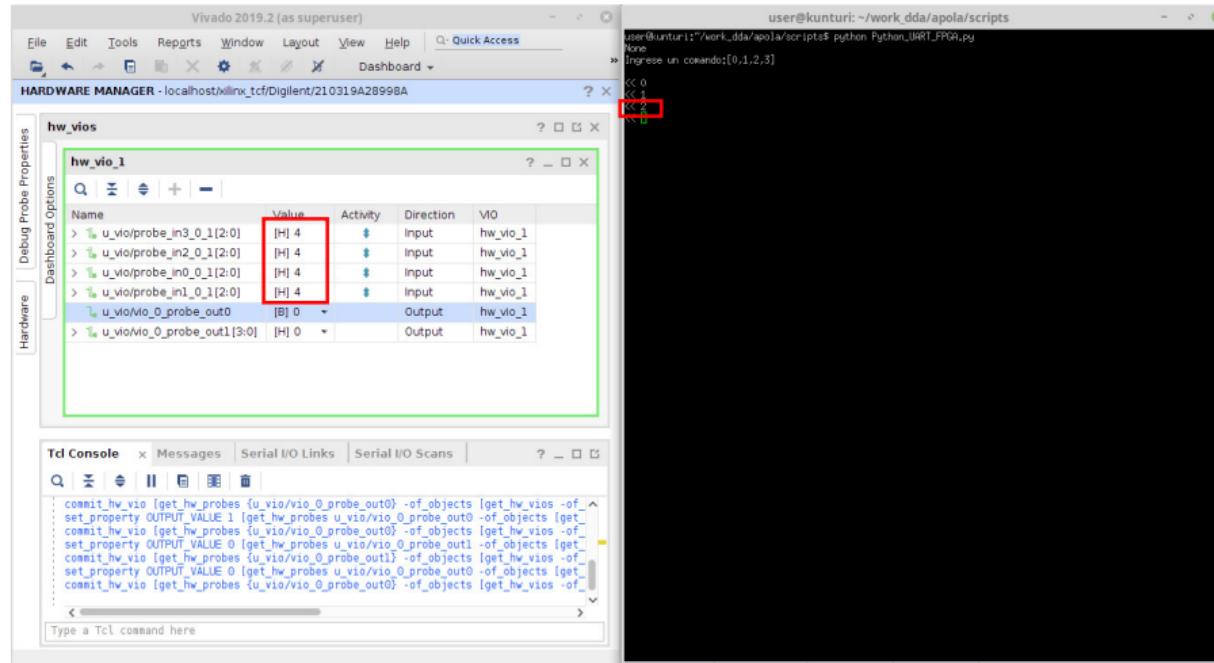
Elegimos 0 y activa el color Rojo (1).

# VIO y Python



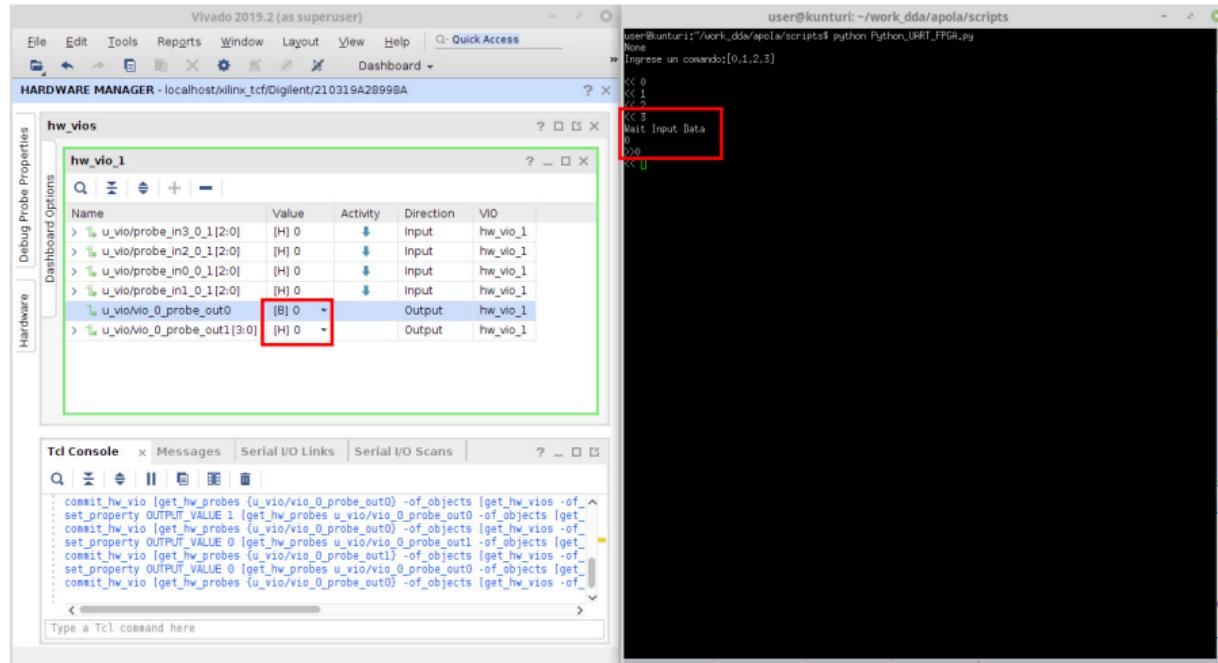
Elegimos 1 y activa el color Verde (2).

# VIO y Python



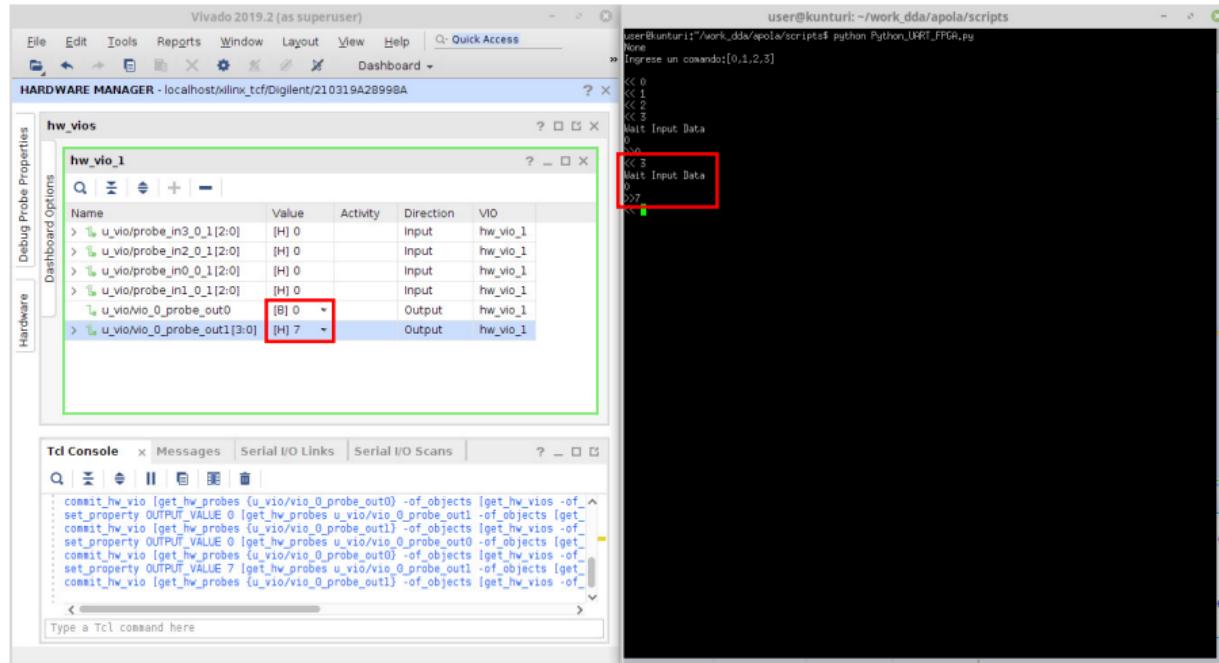
Elegimos 2 y activa el color Azul (4).

# VIO y Python



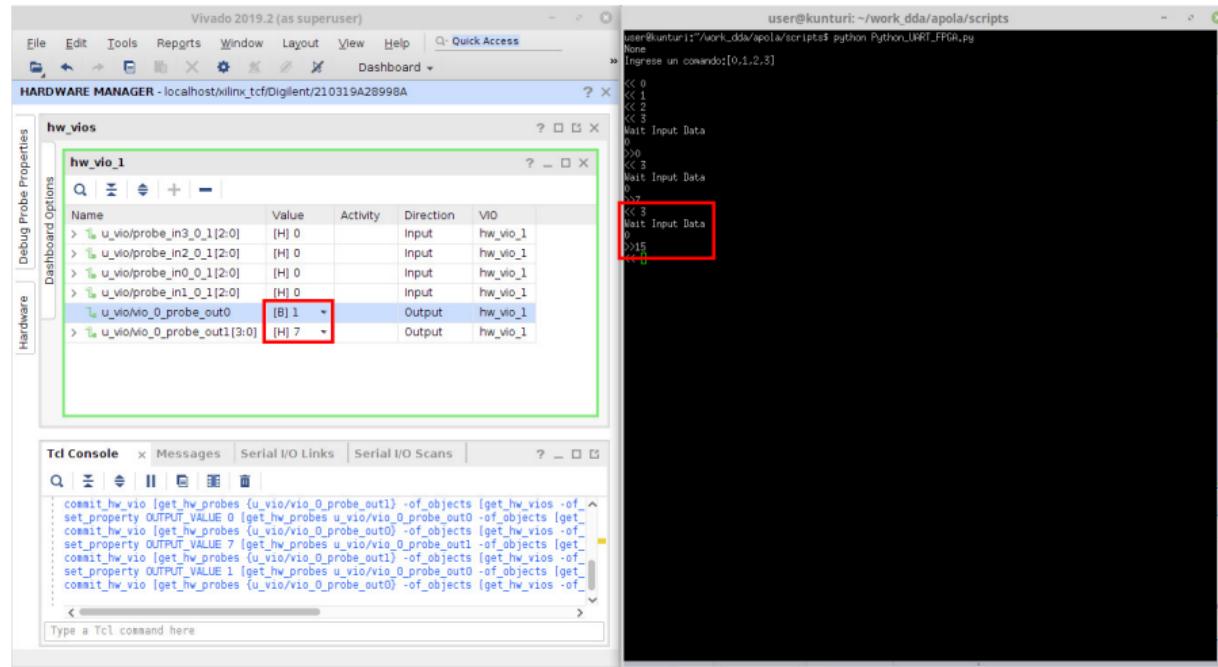
Elegimos 3 para leer el estado de los Switch. Con la salida out0: 0 habilitamos el control desde el VIO y el python tiene que leer el estado de la salida out1 (Python >> 0).

# VIO y Python



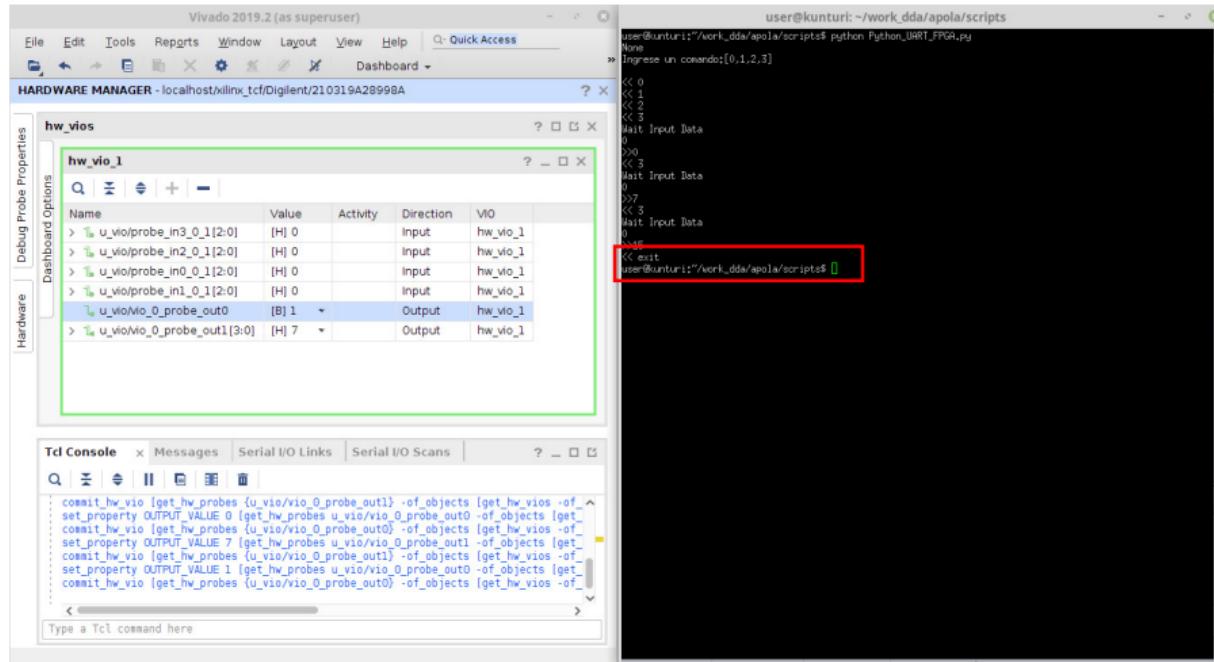
Elegimos 3 para leer el estado de los Switch. Con la salida out0: 0 habilitamos el control desde el VIO y el python tiene que leer el estado de la salida out1 (Python >> 7).

# VIO y Python



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

# VIO y Python



*Salimos del script.*

# Cierre de Sesiones

## Cerrando el Tunel

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

The image displays two terminal windows side-by-side. Both windows have a title bar 'user@kunturi: ~/work\_dda/apola/scripts' and a close button 'x'. The left terminal window shows the user navigating through a script to release a USB port. The right terminal window shows the user closing two tunnel sessions.

**Left Terminal Window:**

```
* 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
```

**Right Terminal Window:**

```
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 tuneles abiertos.