

# **Construcción de un sistema de procesamiento de video utilizando Vivado y Zybo**

**Departamento de Computación, Facultad de Ciencias Exactas y Naturales, UBA**

**Dra. Patricia Borensztein <[patricia@dc.uba.ar](mailto:patricia@dc.uba.ar)>**

**Miguel Ángel García <[miguel.garcia@gmail.com](mailto:miguel.garcia@gmail.com)>**

**Agosto 2015**

# Introducción

Este tutorial explica el desarrollo de un sistema sobre la placa **Zybo**, de Digilent, para la recepción de video vía ethernet, calcular su negativo y reproducir el mismo en un monitor VGA, utilizando la suite de desarrollo **Vivado** de Xilinx.

## Objetivos

- Comprender el flujo de trabajo para la construcción de sistemas que hacen uso de co-diseño hardware/software utilizando Vivado.
- Crear un sistema de procesamiento utilizando Vivado IP Integrator.
- Programar el software que correrá en el sistema de procesamiento utilizando Vivado SDK.
- Crear un IP para mejorar el rendimiento del sistema utilizando Vivado HLS.

## El diseño

El diseño consiste en un sistema que recibe imágenes de video via TCP desde una computadora conectada a la placa Zybo por Ethernet y muestra sus negativos, en un monitor conectado al puerto VGA.

Hace uso de PL para instanciar el controlador de display, que es un IP provisto por Digilent, y un controlador VDMA para enviar las imágenes de la RAM central al controlador de display.

Finalmente integraremos al diseño un IP propio para el procesamiento de la imagen.

## Procedimiento

Este tutorial está separado en pasos que describen detalladamente las tareas necesarias para la construcción del sistema.

## Flujo de diseño

1. Crear proyecto en Vivado IP Integrator
2. Exportar el diseño a Vivado SDK, crear Board Support Package y Aplicación
3. Correr la aplicación en Zybo
4. Crear proyecto Vivado HLS, correr test en C, sintetizar el diseño y exportar RTL
5. Importar IP en Vivado e Integrar IP al diseño
6. Exportar a SDK, modificar aplicación para usar el nuevo IP

## 1 - Crear proyecto en Vivado IP Integrator

### 1.1 - Crear un nuevo proyecto en Vivado IP Integrator apuntando a la plataforma Zynq

1.1.1 – Hacer click en el botón “Create New Project” y hacer click en “Next” en la ventana que aparece.

1.1.2 – Ingresar un nombre para el proyecto y la ruta donde se guardara, luego hacer click en “Next” y en la siguiente ventana hacer click nuevamente en “Next”

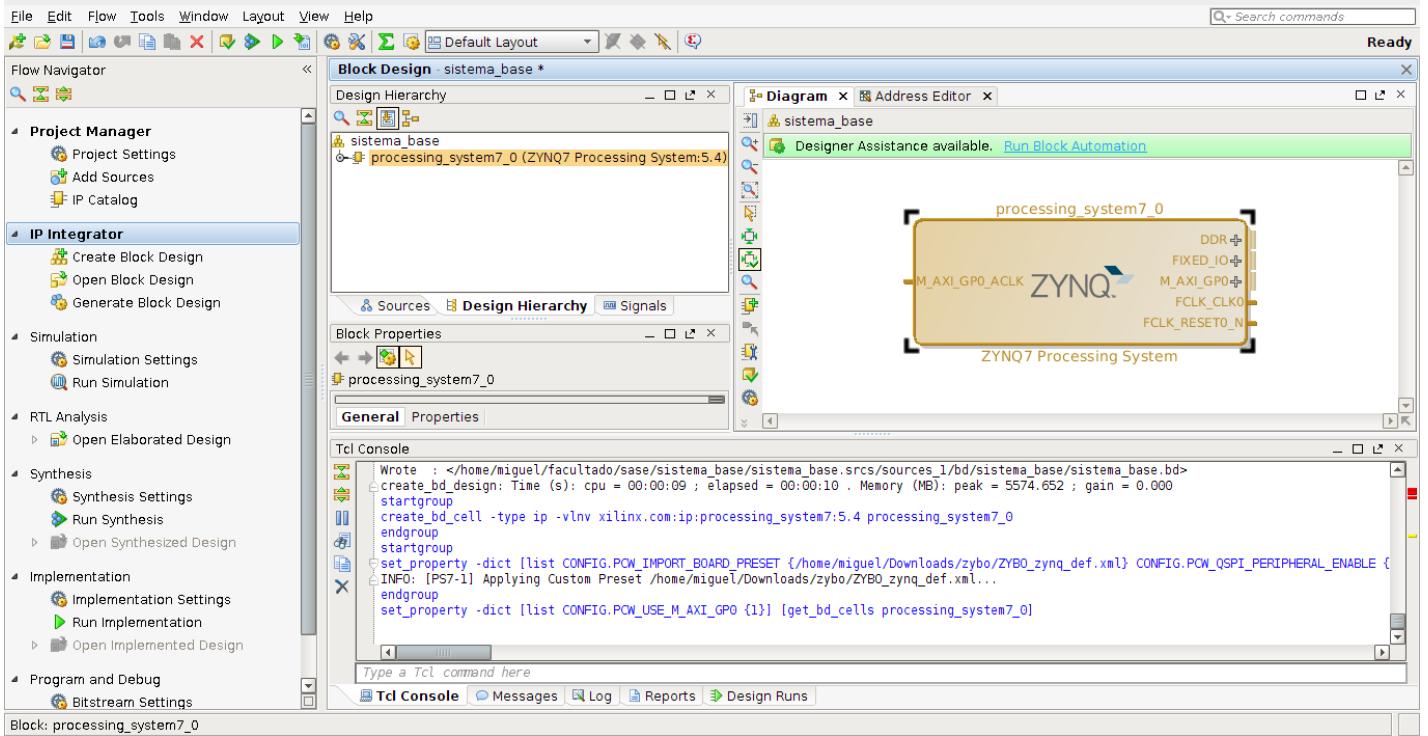
1.1.3 – Seleccionar Part xc7z010clg400-1 (es el chip Zynq que utiliza la placa Zybo) y hacer click en “Next”

1.1.4 – Verificar datos y si todo está bien hacer click en “Finish”

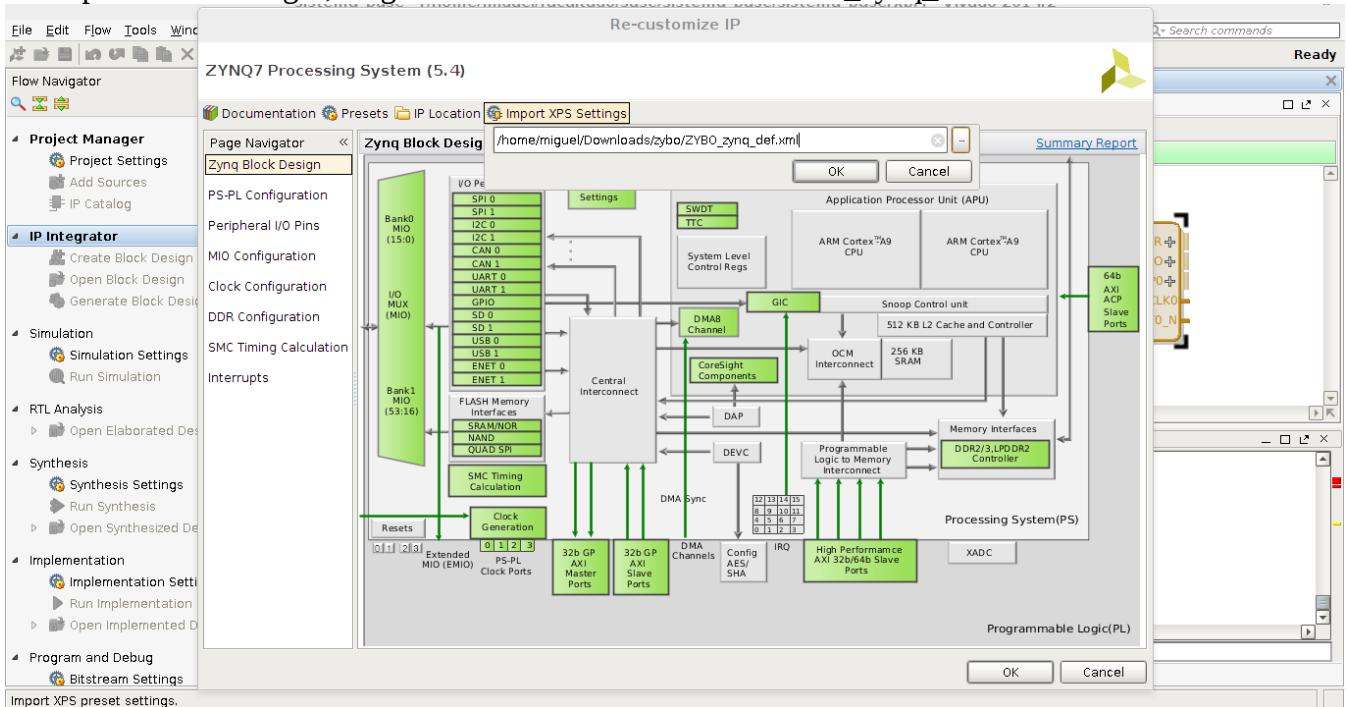
## 1.2 – Crear Block Design, donde integraremos Zynq con otros IPs, los configuraremos y definiremos los puertos externos.

1.2.1 – En el flow navigator (menú de la izquierda) hacer click en “Create Block Design”, darle al diseño el nombre “sistema\_base” y hacer click en “Ok”

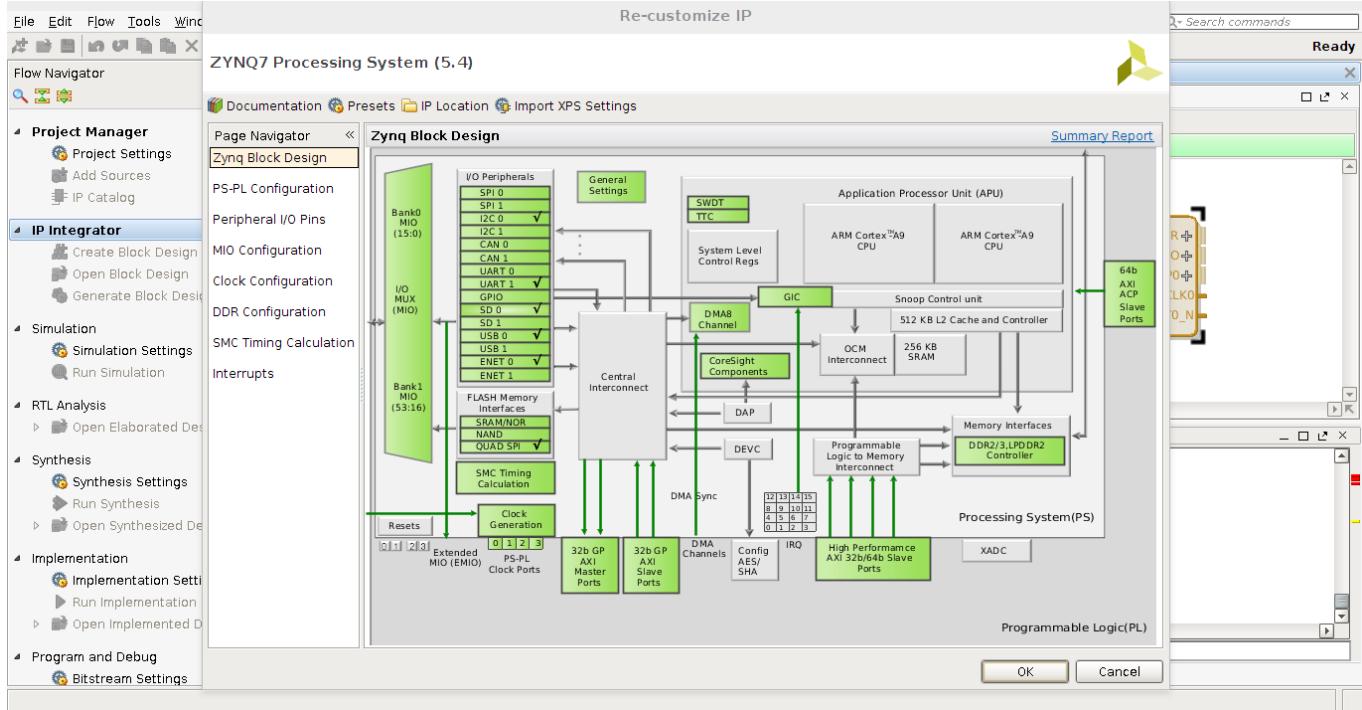
1.2.2 – Agregar al diseño el IP “ZYNQ7 Processing System”



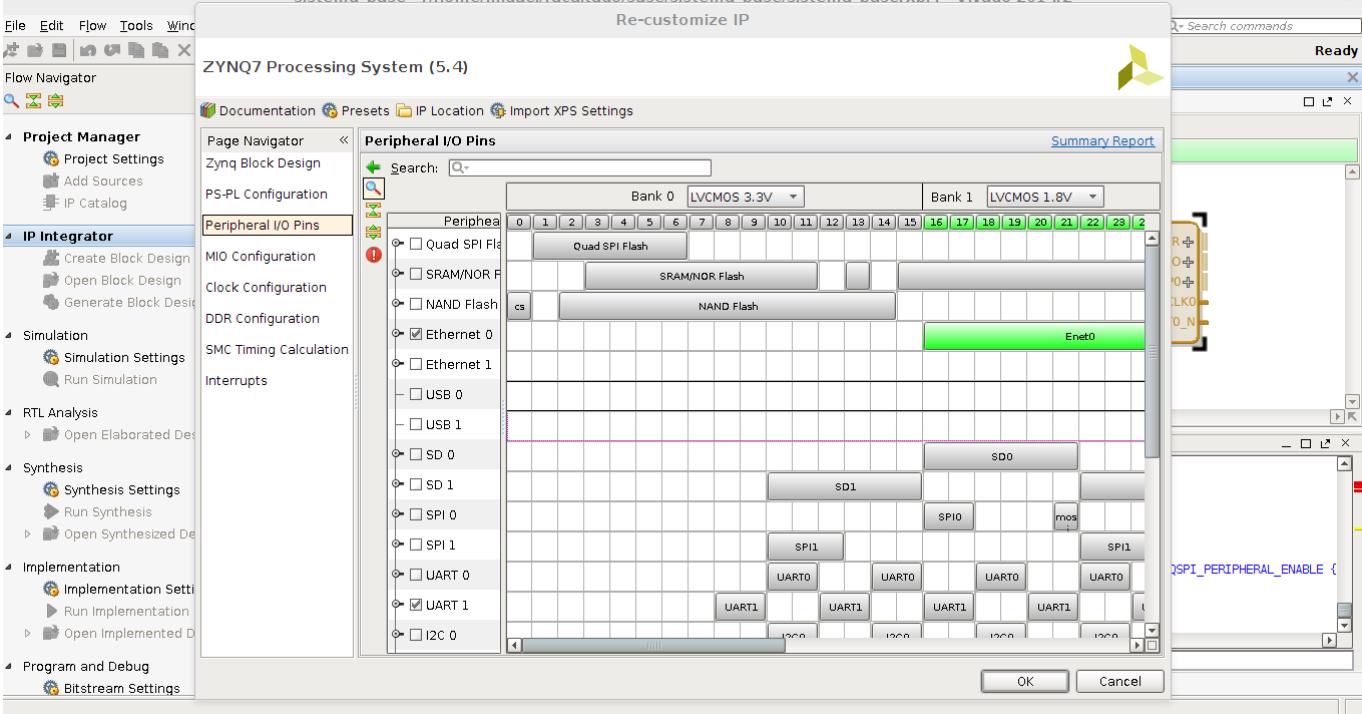
1.2.3 – Configurar el IP importando el archivo de definiciones provisto por Digilent, haciendo click derecho en el IP, seleccionar la opción “Customize Block” y en la ventana que aparece seleccionar “Import XPS Settings”, luego indicar la ruta del archivo ZYBO\_zynq\_def.xml.



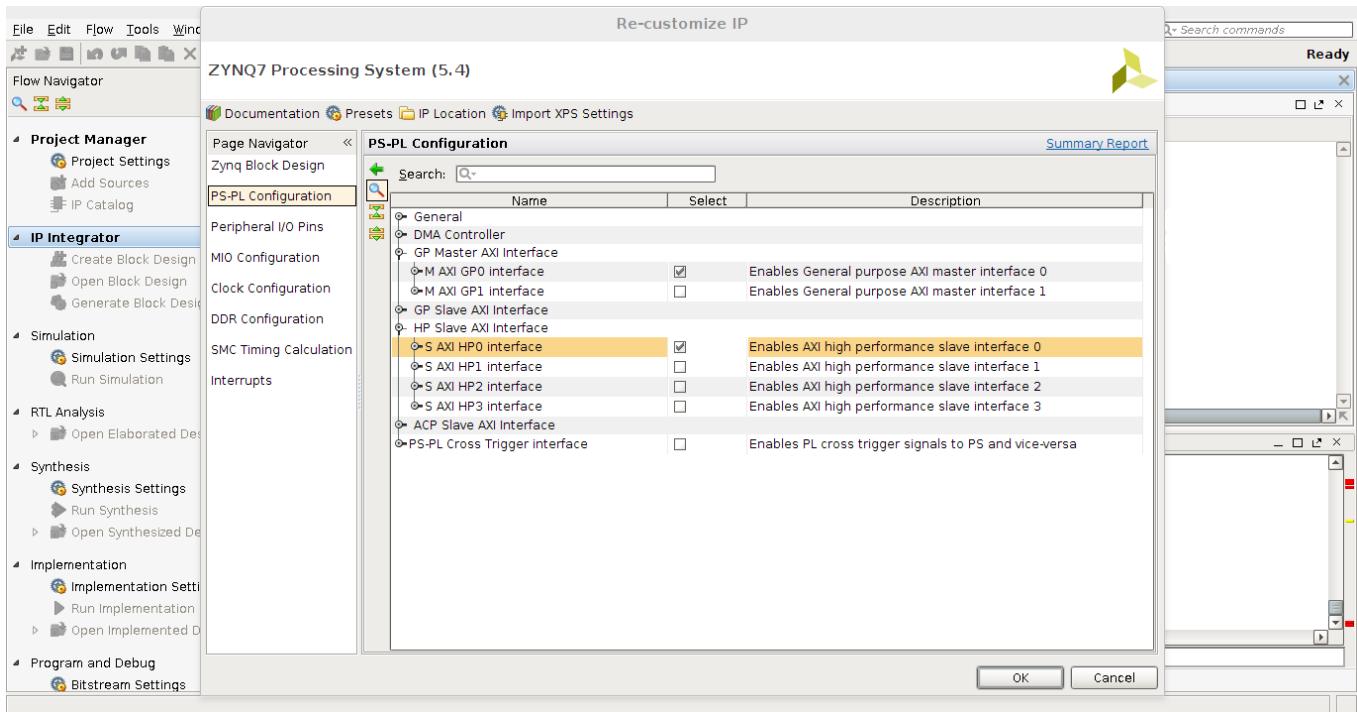
Luego hacer click en “Ok”



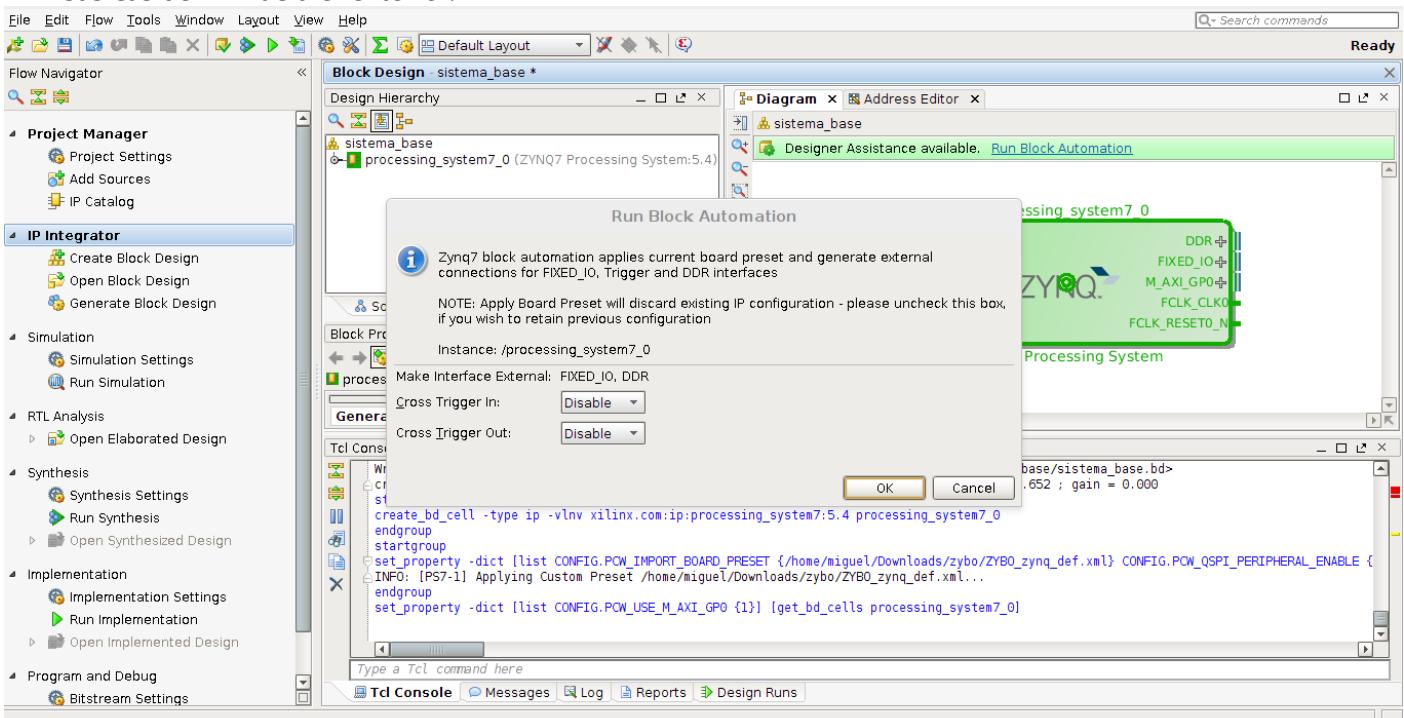
**1.2.4 – Deshabilitar las interfaces que no utilizaremos en el proyecto, hacer click en “Peripheral I/O Pins” y destildar todas las opciones, excepto “Ethernet0” y “UART1”**



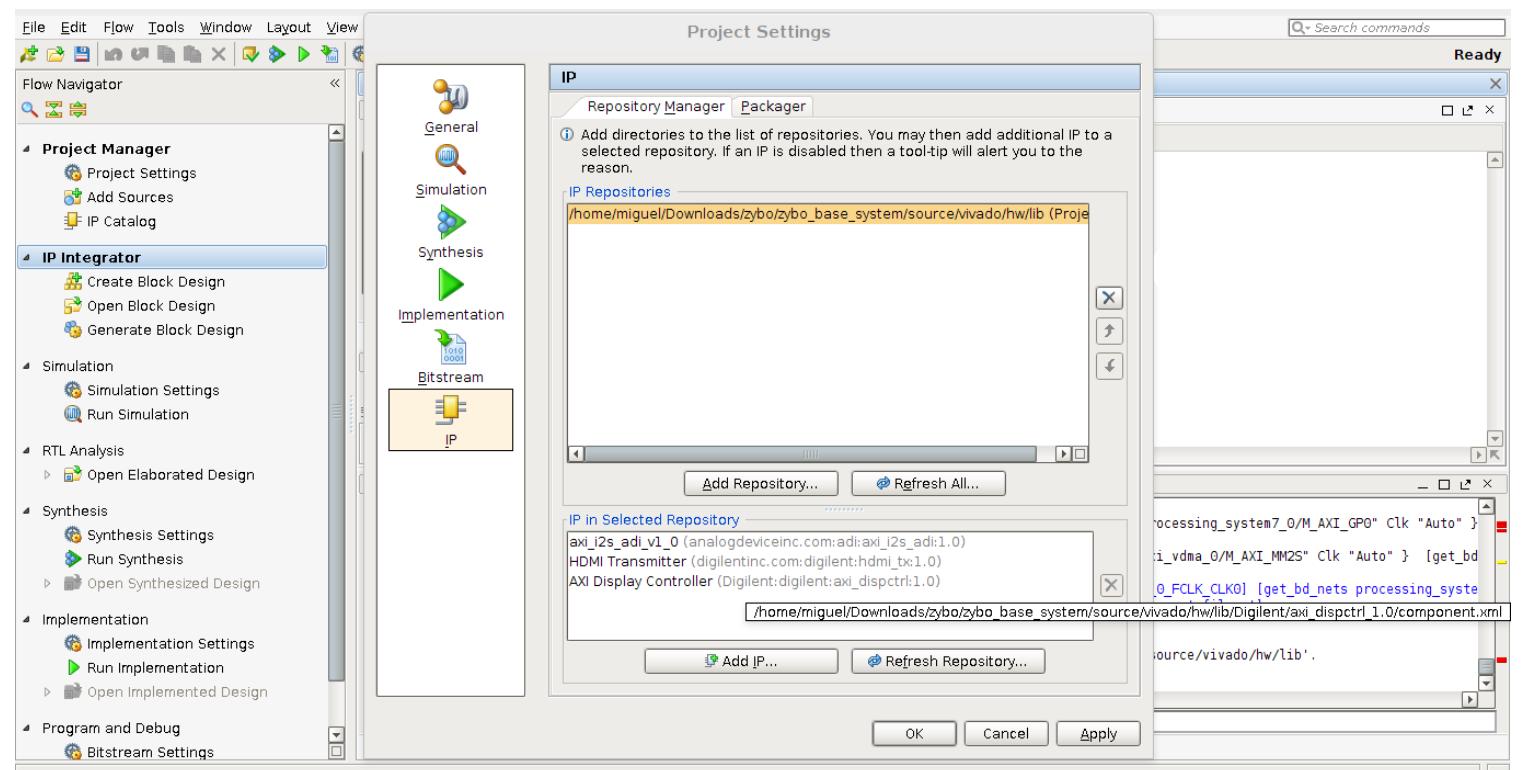
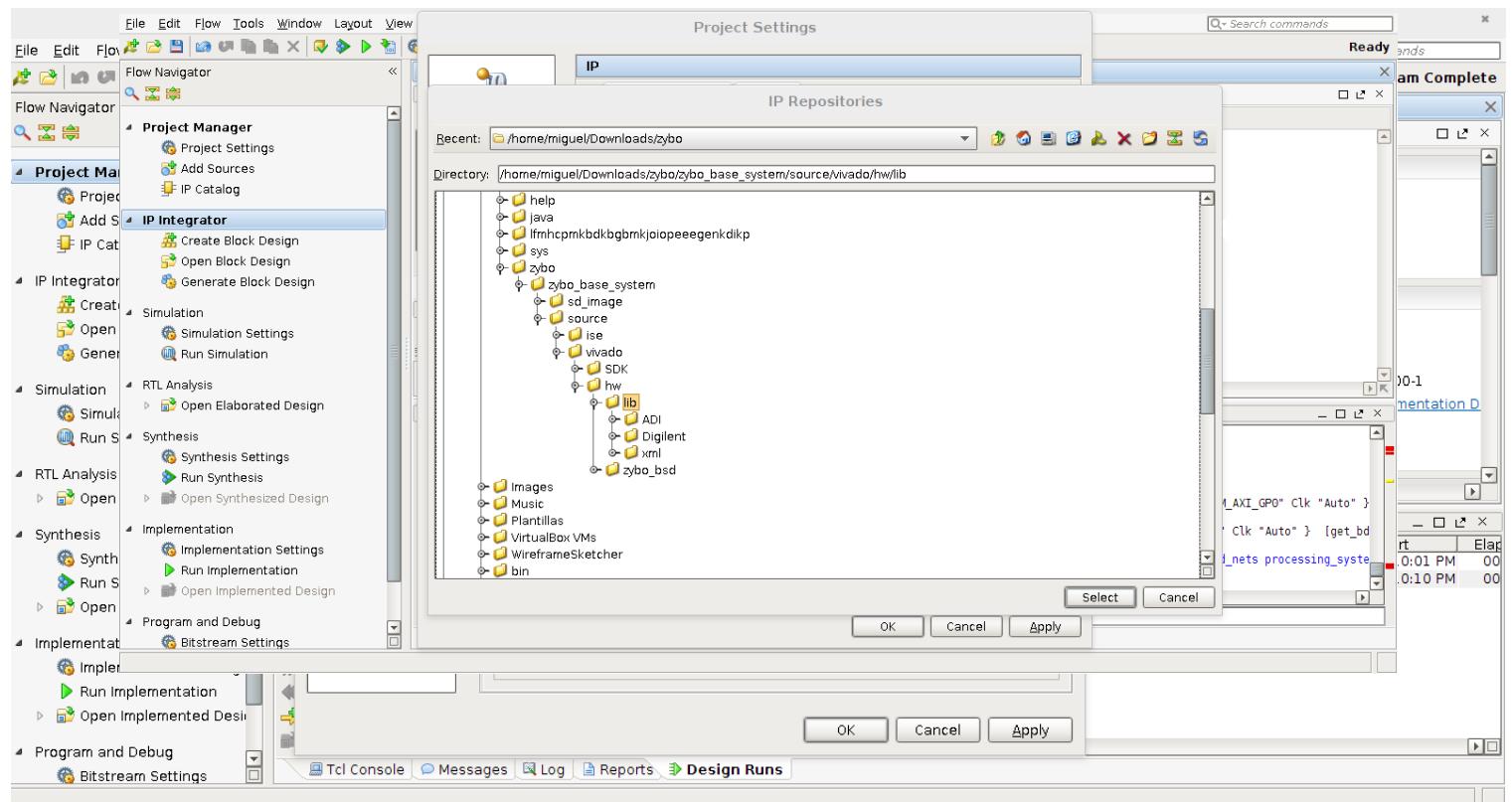
**1.2.5 – Habilitar interfaz HP de Zynq haciendo click en “PS/PL Configuration” y luego tildar la opción “S AXI HP0 Interface”**



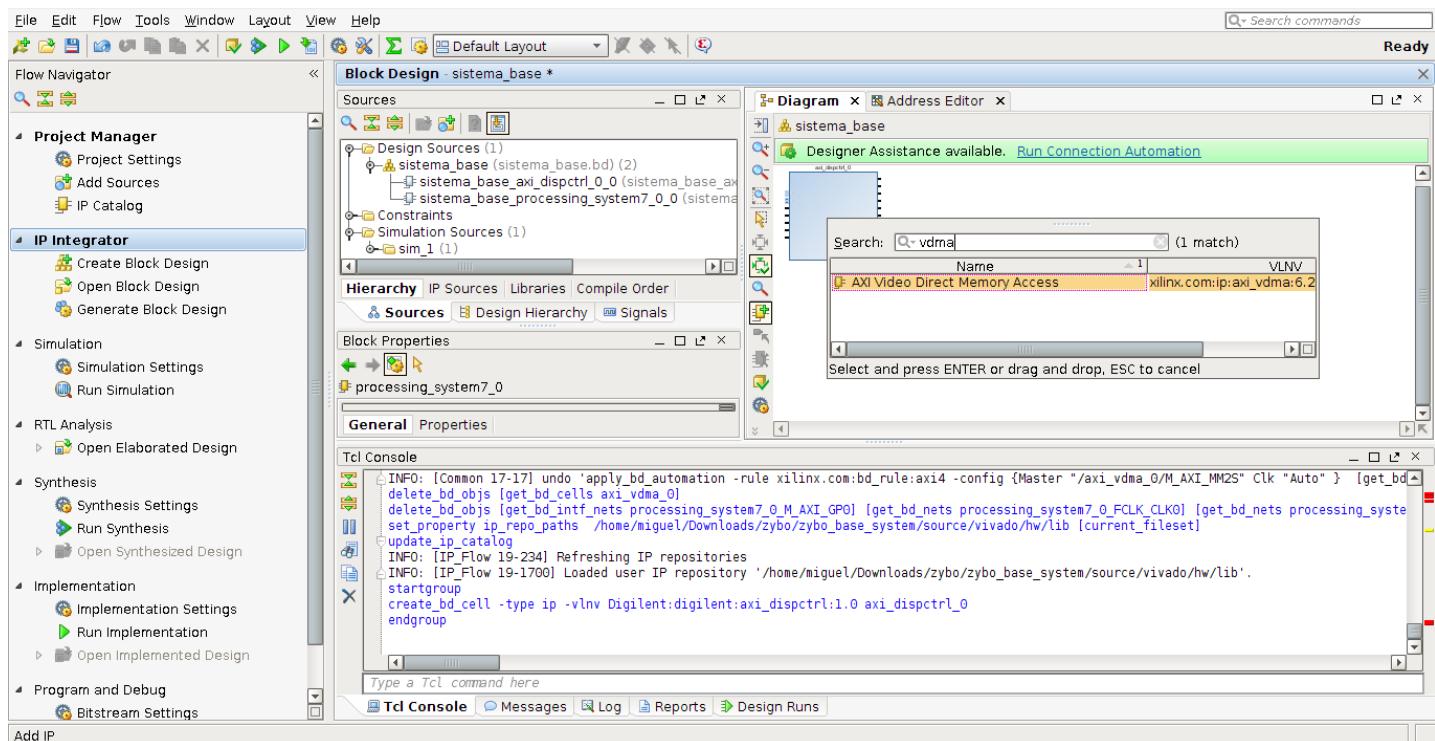
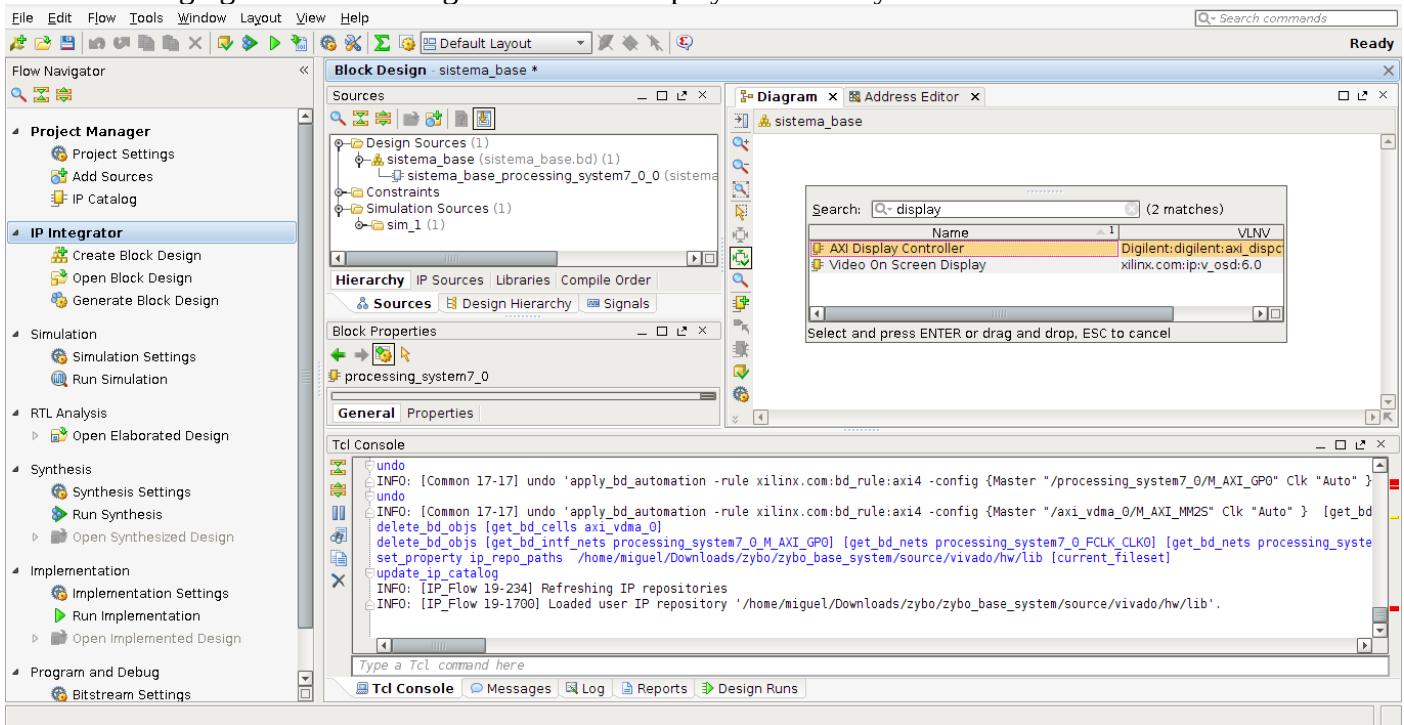
**1.2.6 – Hacer click en “Run Block Automation” y en el botón “Ok” para crear las conexiones básicas del IP hacia el exterior.**

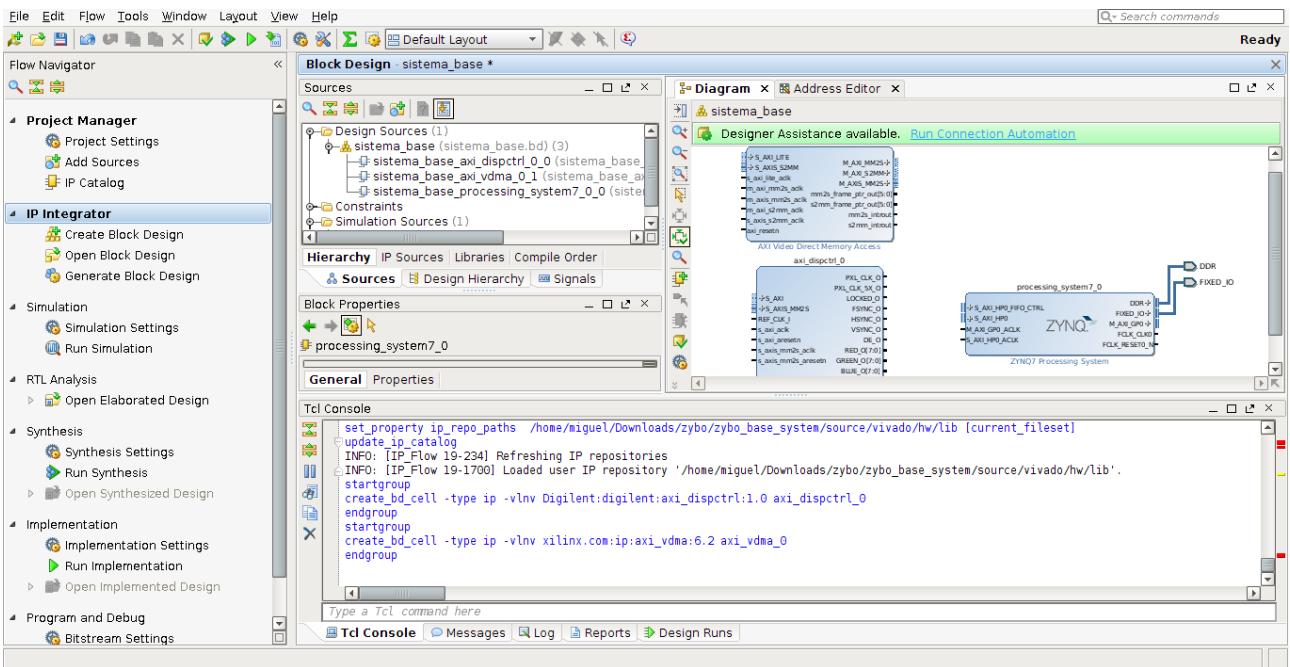


**1.2.7 – Importar el IP Display Controller, de Digilent: En el menú ir a Tools → Project Settings, en la ventana que se abre seleccionar “IP”. Hacer click en “Add Repository” e introducir la ruta de los IPs provistos por Digilent, hacer click en “Select” y luego en “Ok”.**

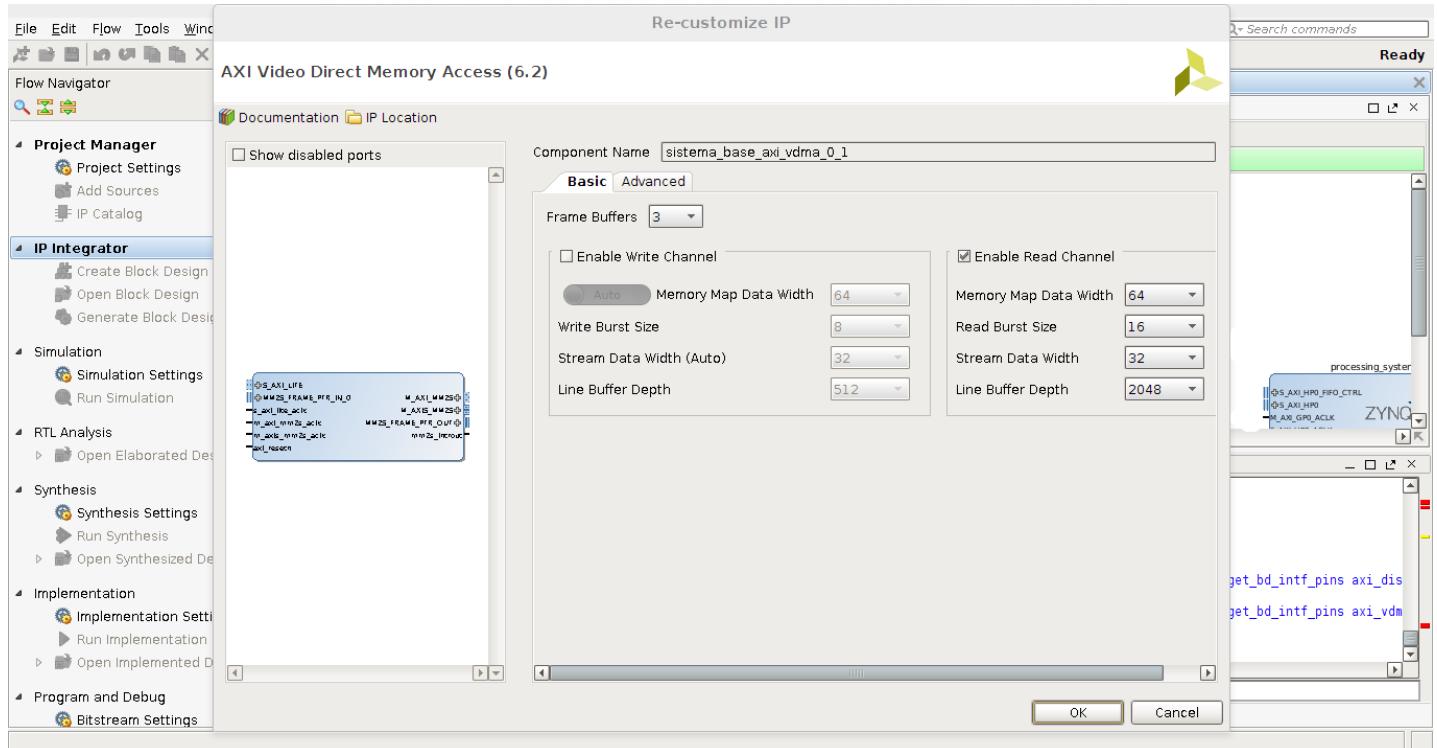


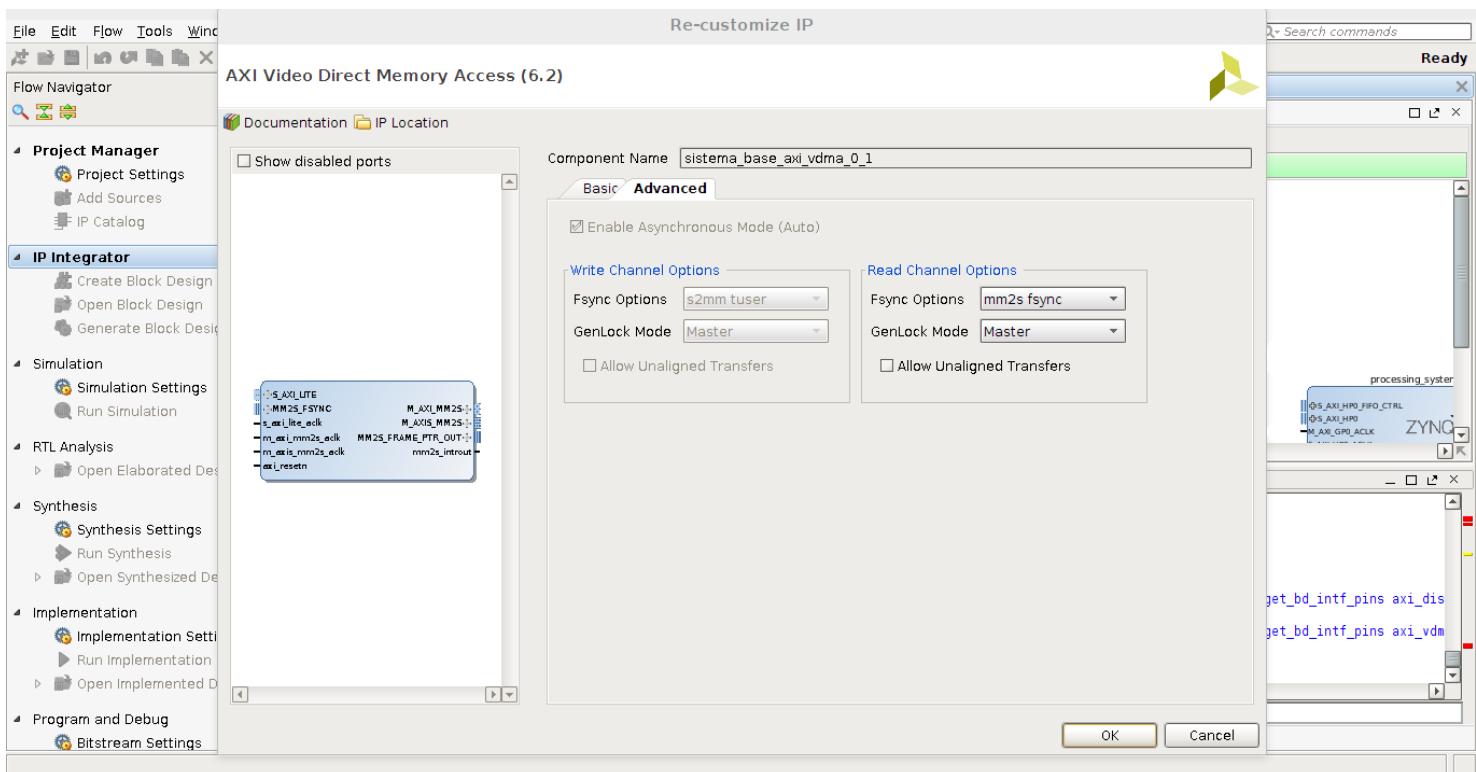
## 1.2.8 – Agregar al Block Design los IP AXI Display Controller y AXI VDMA



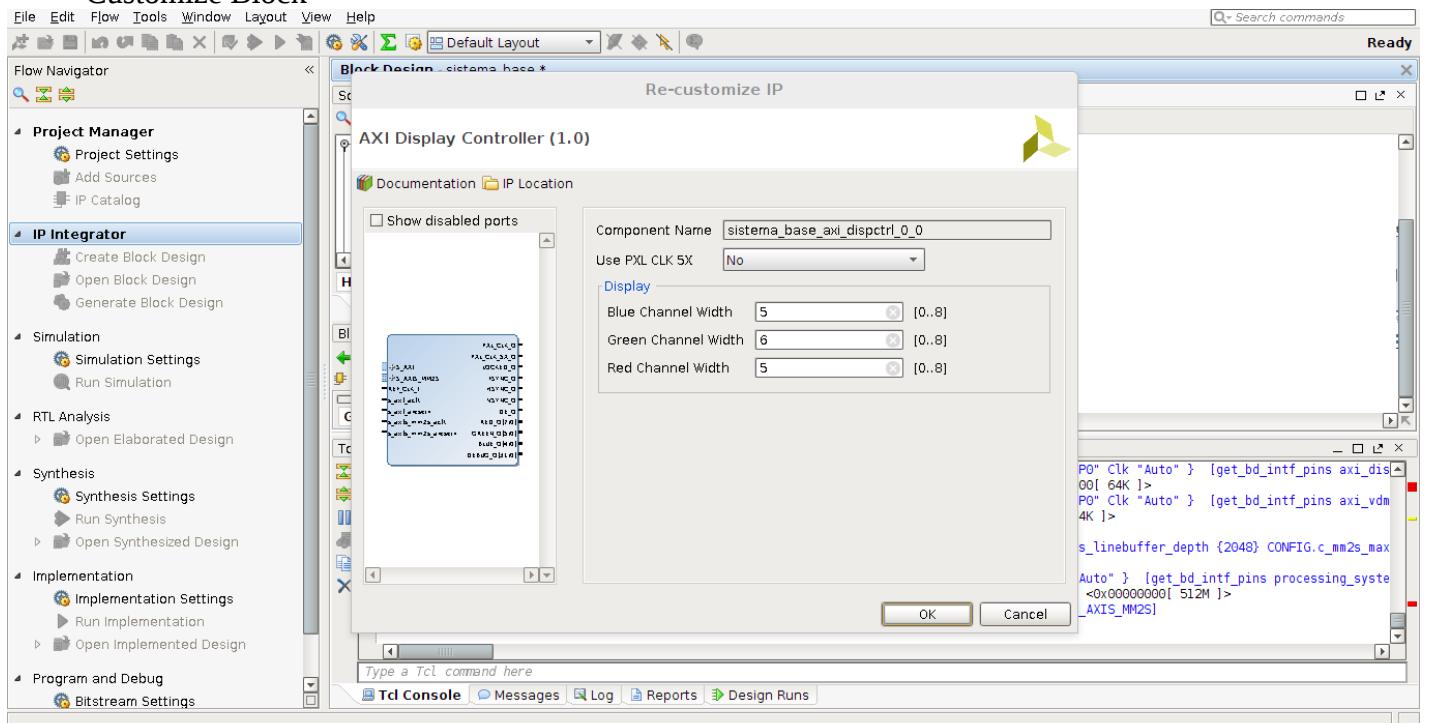


### 1.2.9 – Configurar el IP AXI VDMA, hacer click derecho sobre el mismo y seleccionar “Customize Block”

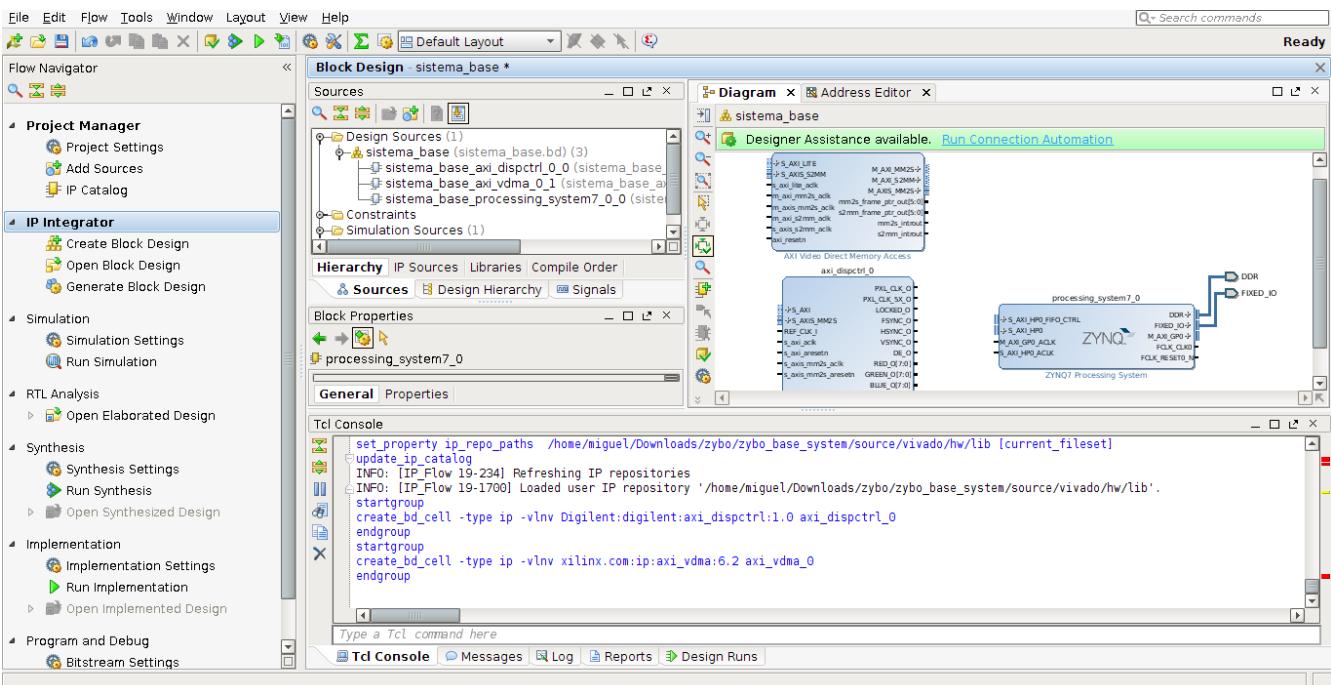




### 1.2.10 – Configurar el IP AXI Display Controller, hacer click derecho sobre el mismo y seleccionar “Customize Block”

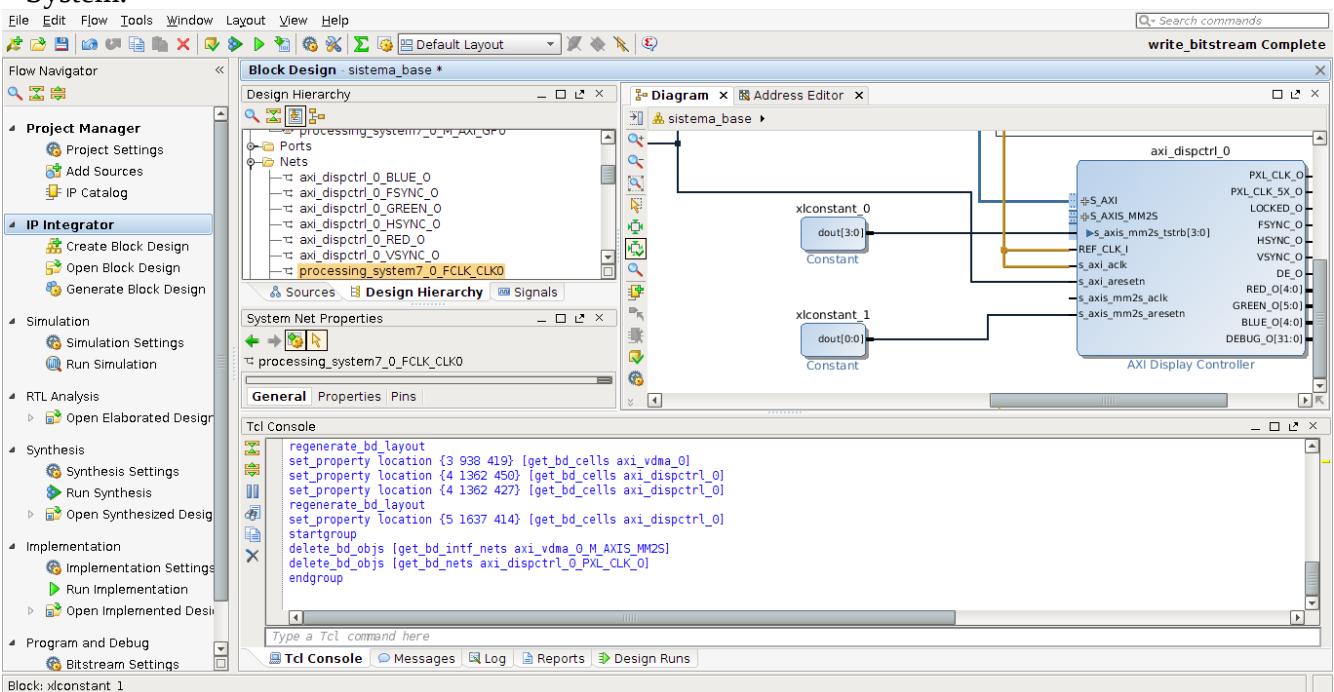


### 1.2.11 – Conectar los IP AXI Display Controller y AXI VDMA al Zynq Processing System, hacer click en “Run Connection Automation”

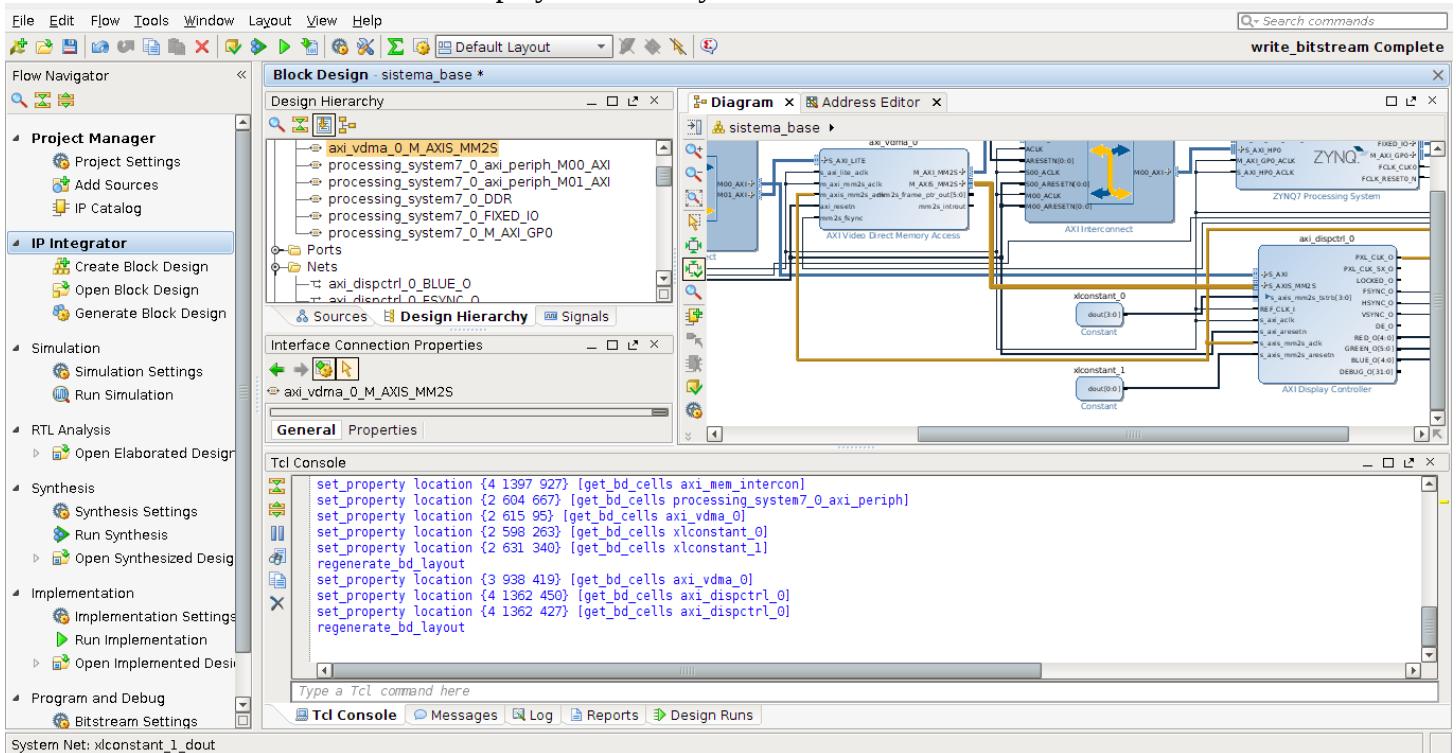


Esta acción hay que realizarla 3 veces, una por cada conexión de los IP al bus AXI GP y otra para conectar el VDMA al AXI HP

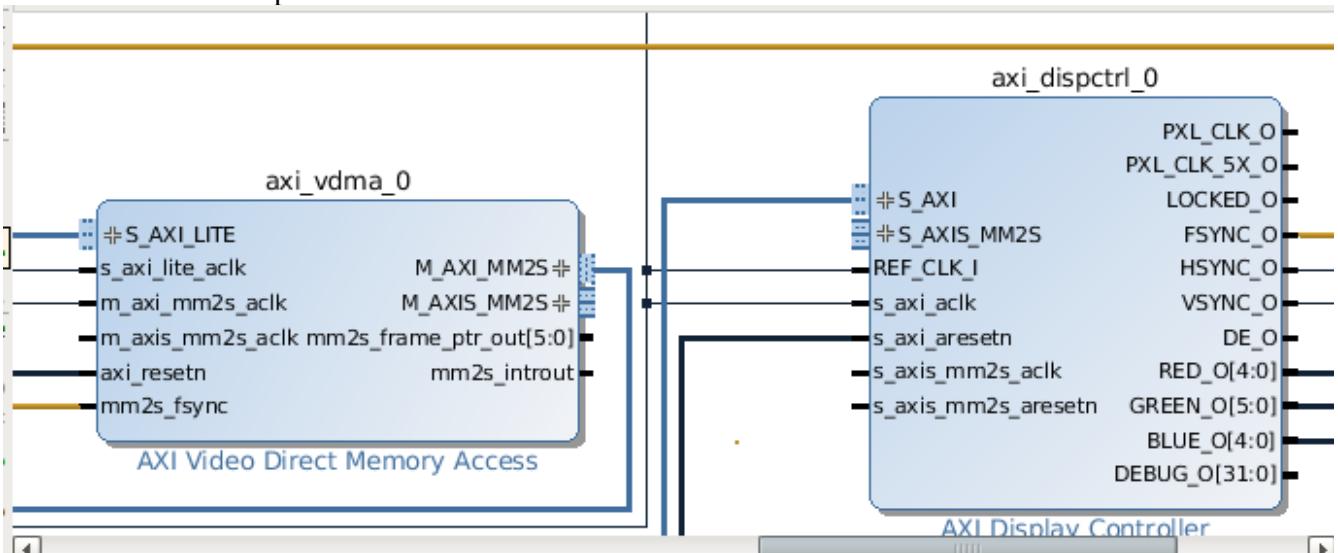
**1.2.12 – Conectar el IP AXI Display Controller al clock de referencia y agregar bloques Constant para fijar los valores de los puertos s\_axis\_mm2s\_trtrb y s\_axis\_mm2s\_aresetn a 1111 y 1 respectivamente. El puerto REF\_CLK\_I conectarlo al puerto FCLK\_CLK0 del Zynq Processing System.**



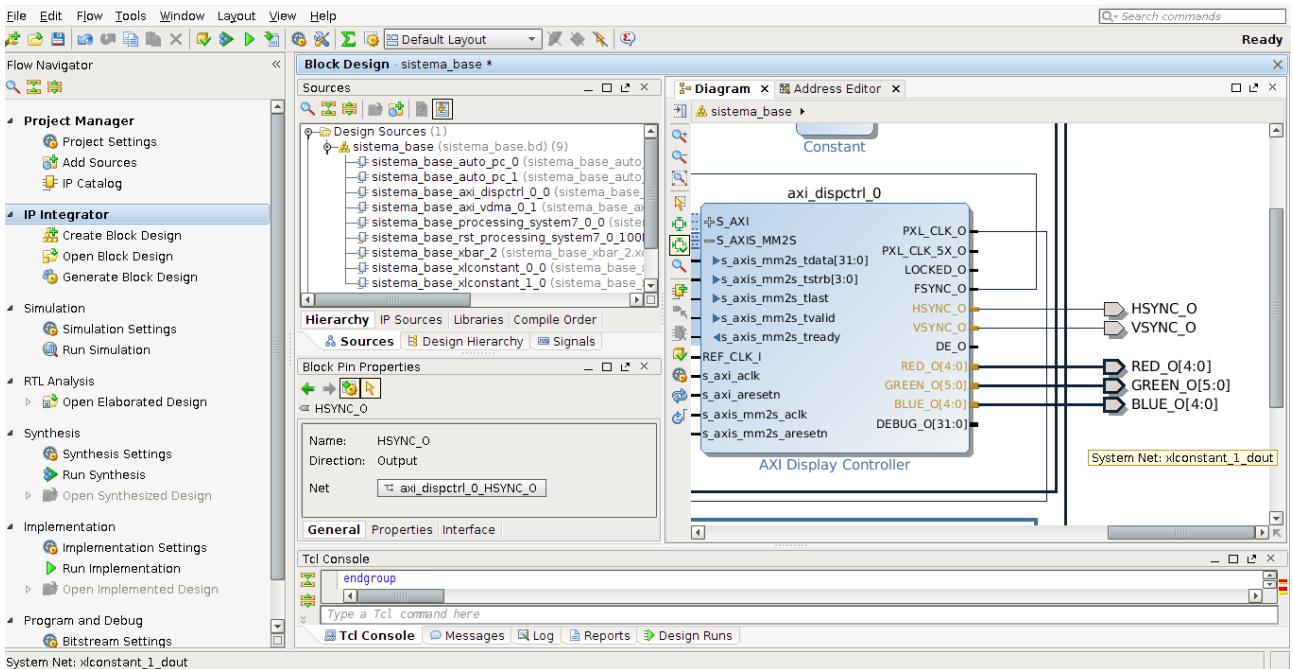
### 1.2.13 – Conectar el IP AXI Display Controller y el IP AXI VDMA entre si.



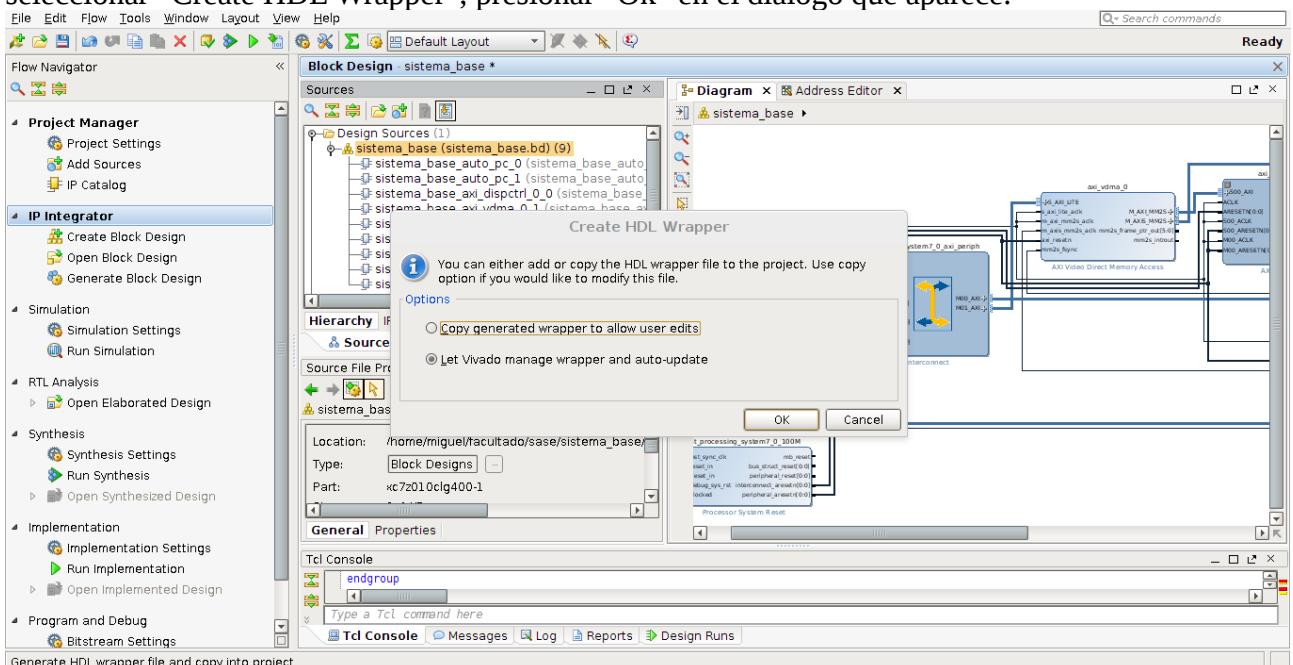
No olvidar conectar el puerto FSYNC



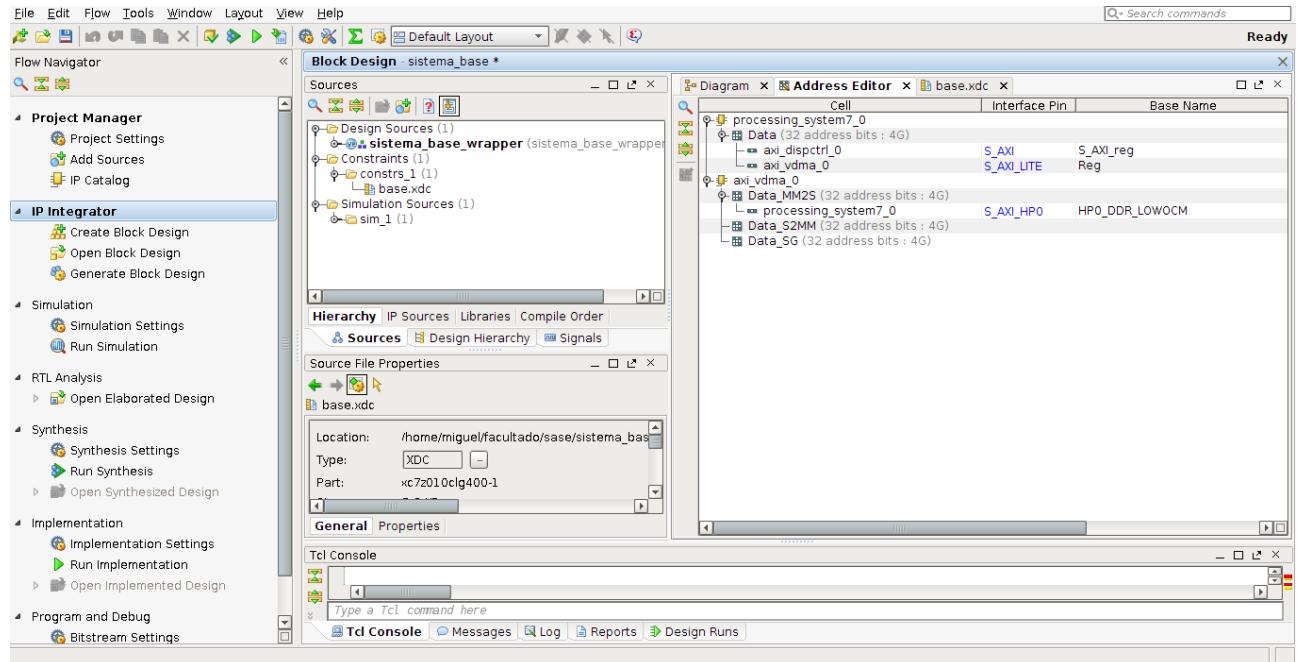
### 1.2.14 – Hacer externos los puertos VGA del AXI Display Controller



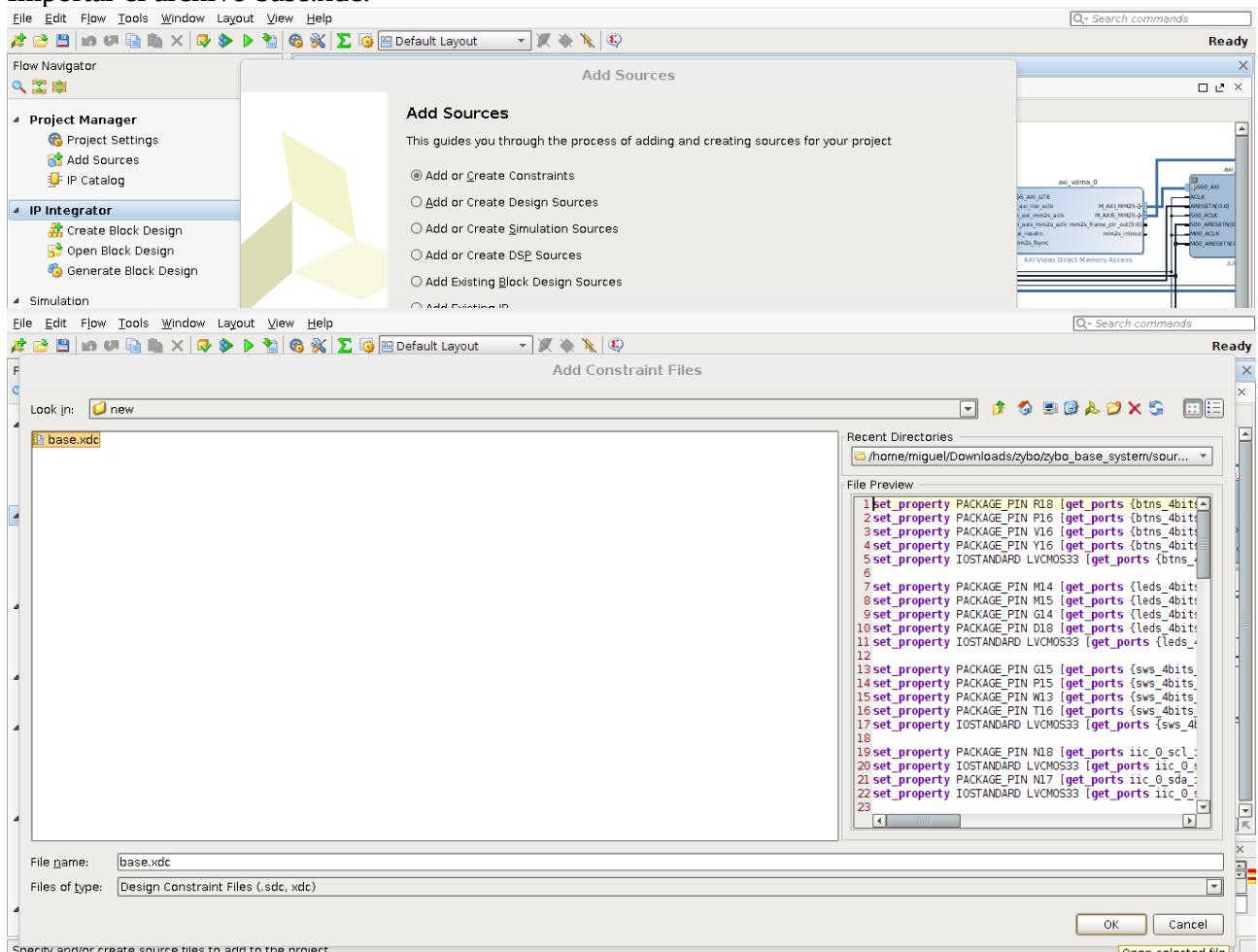
**1.2.15 – Crear wrapper HDL:** en el menú “Sources” hacer click derecho sobre sistema\_base y seleccionar “Create HDL Wrapper”, presionar “Ok” en el dialogo que aparece.



**1.2.16 – Revisar los espacios de direcciones para asegurarse que todos los dispositivos se encuentre mapeados en el espacio de direcciones del Zynq y, además, la memoria principal en el IP VDMA.**  
**Seleccionar la solapa “Address Editor”:**



**1.3 – Definir constraints: Hacer click derecho en “Constraints” y seleccionar “Add Sources”. Importar el archivo base.xdc.**

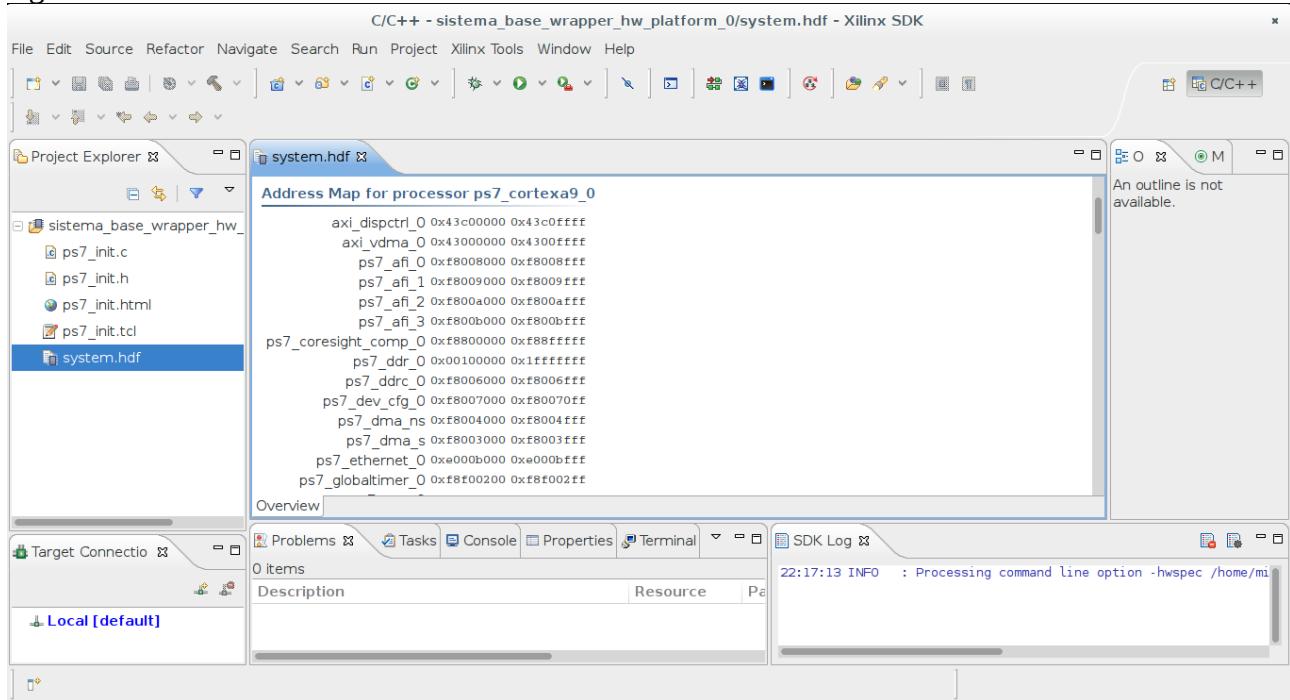


**1.4** – Generar bitstream haciendo click en “Generate Bitstream”, en el menú Flow Navigator.

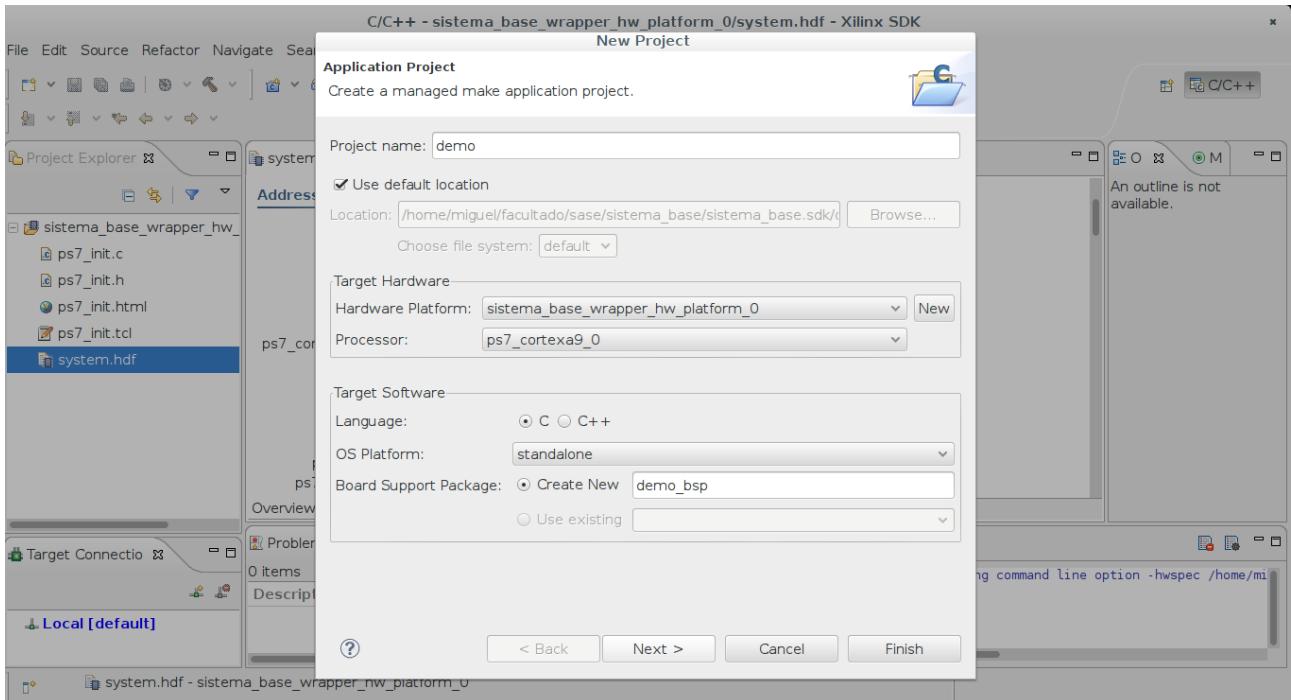
## 2 - Exportar el diseño a Vivado SDK, crear Board Support Package y Aplicación

**2.1** – En Vivado IP Integrator seleccionar la opción de menú File → Export → Export Hardware y hacer click en “Ok” en el dialogo emergente.

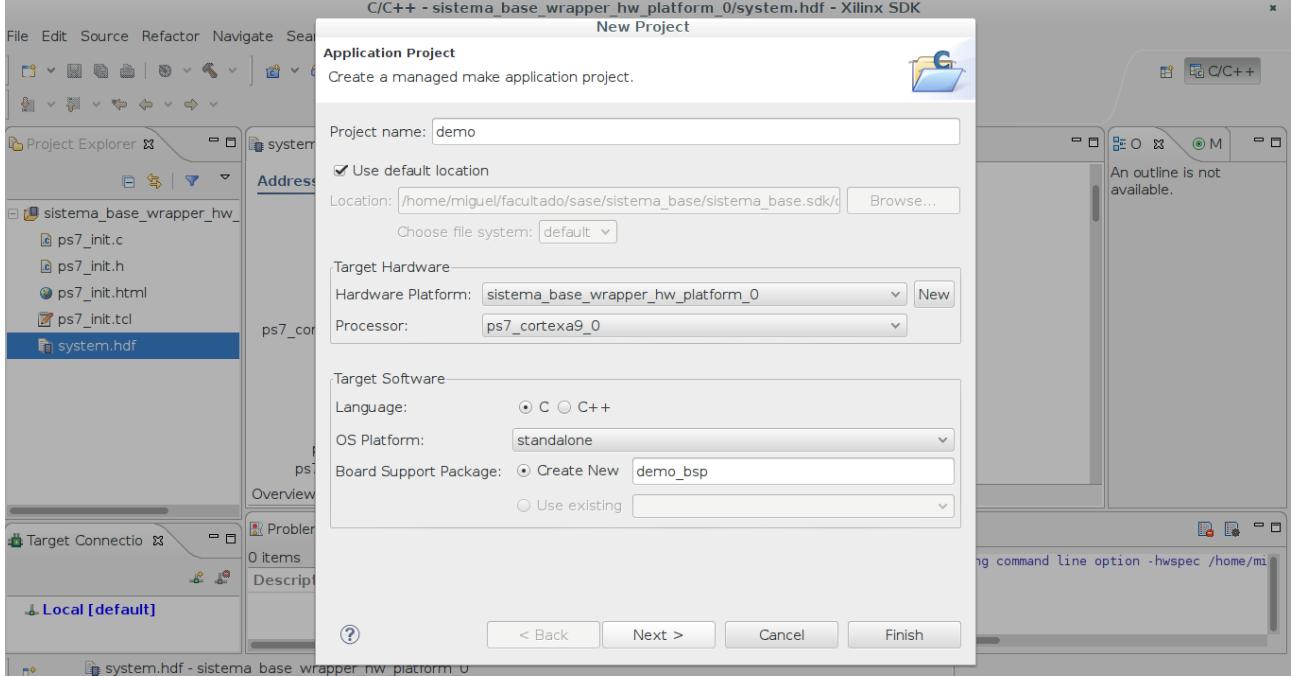
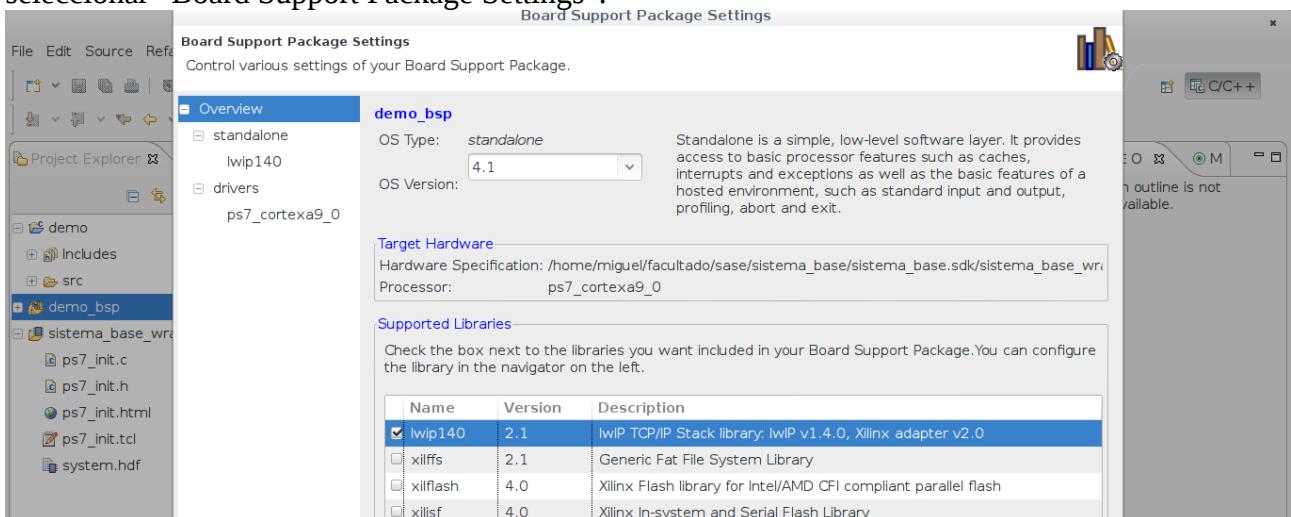
**2.2** – Lanzar Vivado SDK, En Vivado IP Integrator seleccionar la opción de menú File → Launch SDK, hacer click en “Ok” en el dialogo emergente. A continuación se abrirá Vivado SDK con la siguiente ventana:



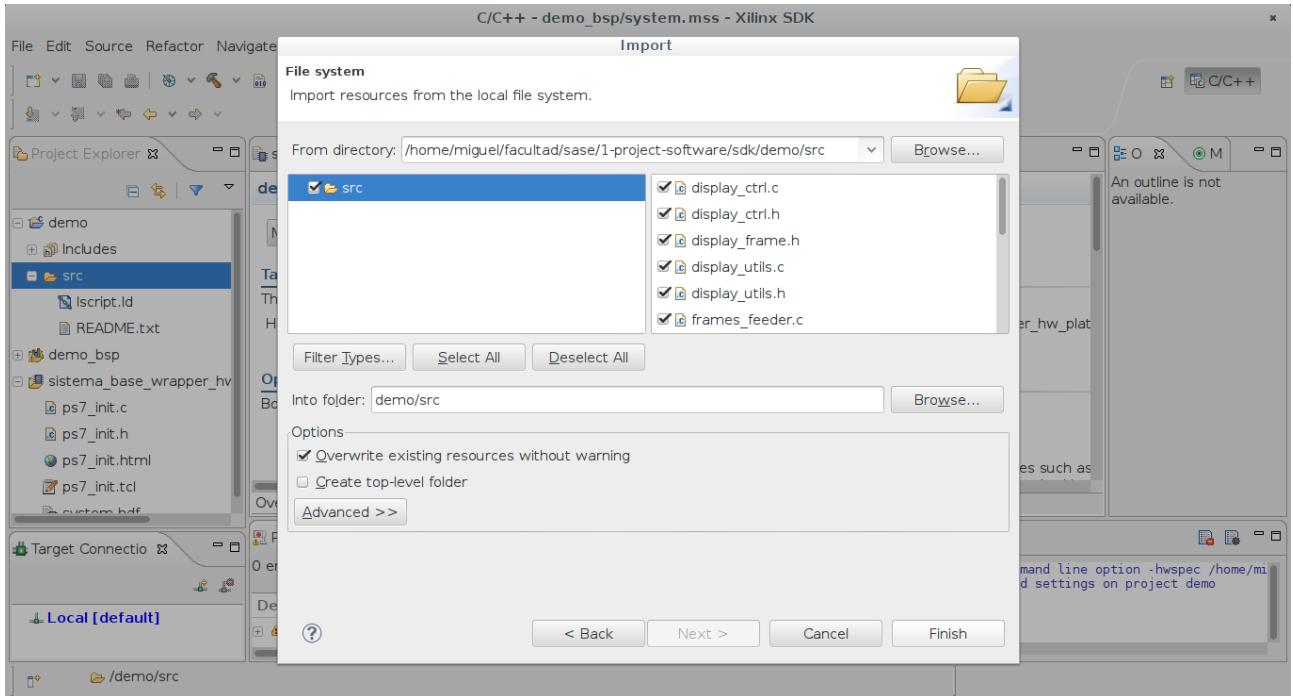
**2.3** – Crear Board Support Package y esqueleto de aplicación: En el menú seleccionar File → New → Application Project. Ingresar el nombre para el proyecto “demo” y presionar “Next”. Luego seleccionar el template “Empty Application” y hacer click en “Finish”



**2.4 – Configurar BSP para agregar soporte TCP/IP: Hacer click derecho sobre demo\_bsp y seleccionar “Board Support Package Settings”.**



**2.5 – Importar el código fuente de la aplicación:** hacer click derecho en “demo” y seleccionar “Import”. En el dialogo que se abre seleccionar General → File System y hacer click en “Next”. Búscar el directorio con el código provisto, luego hacer click en “Select All” y finalmente en “Finish”



## 3 – Correr la aplicación creada en Zybo

**3.1 – Configurar placa de red en la computadora:**

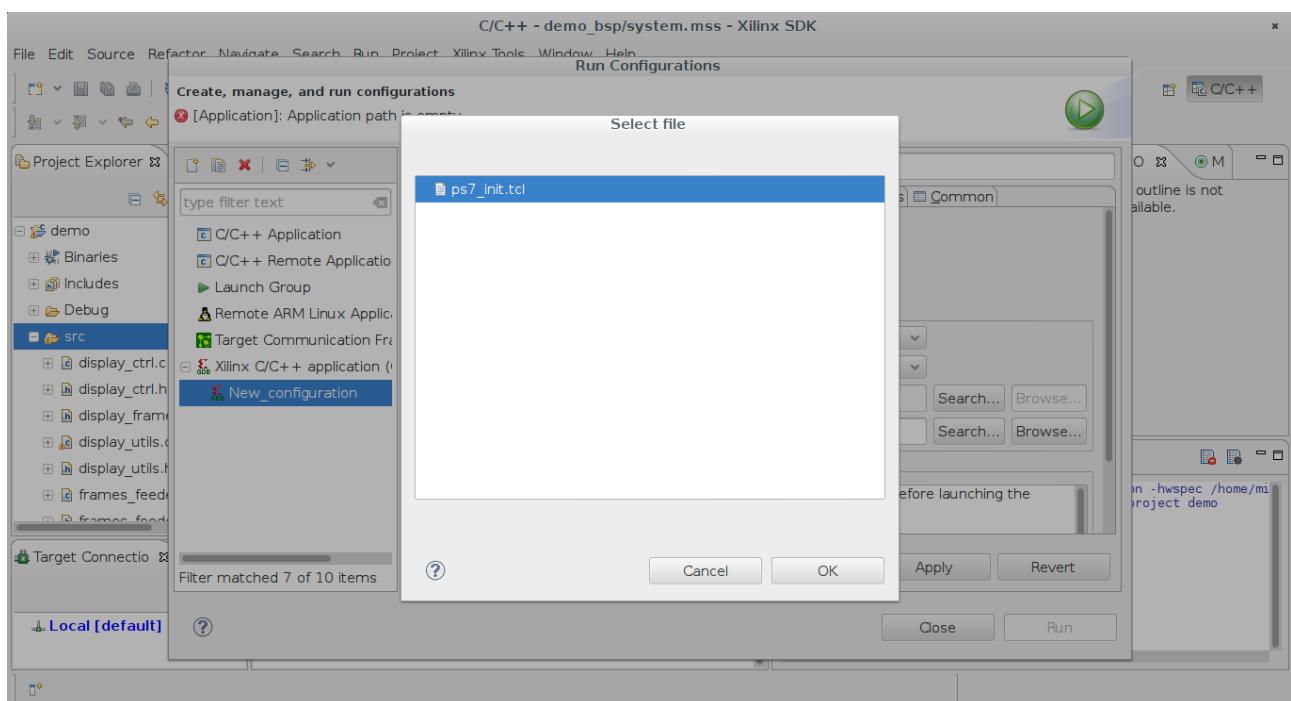
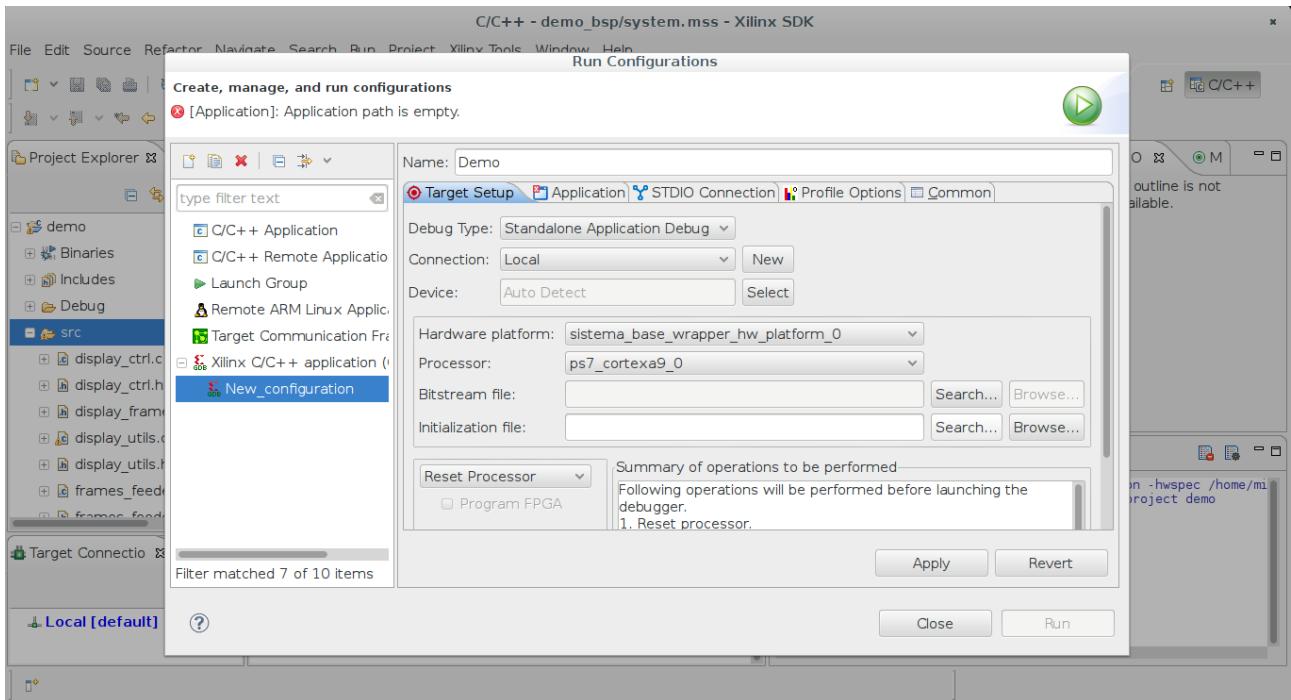
IP estática: 10.0.0.1

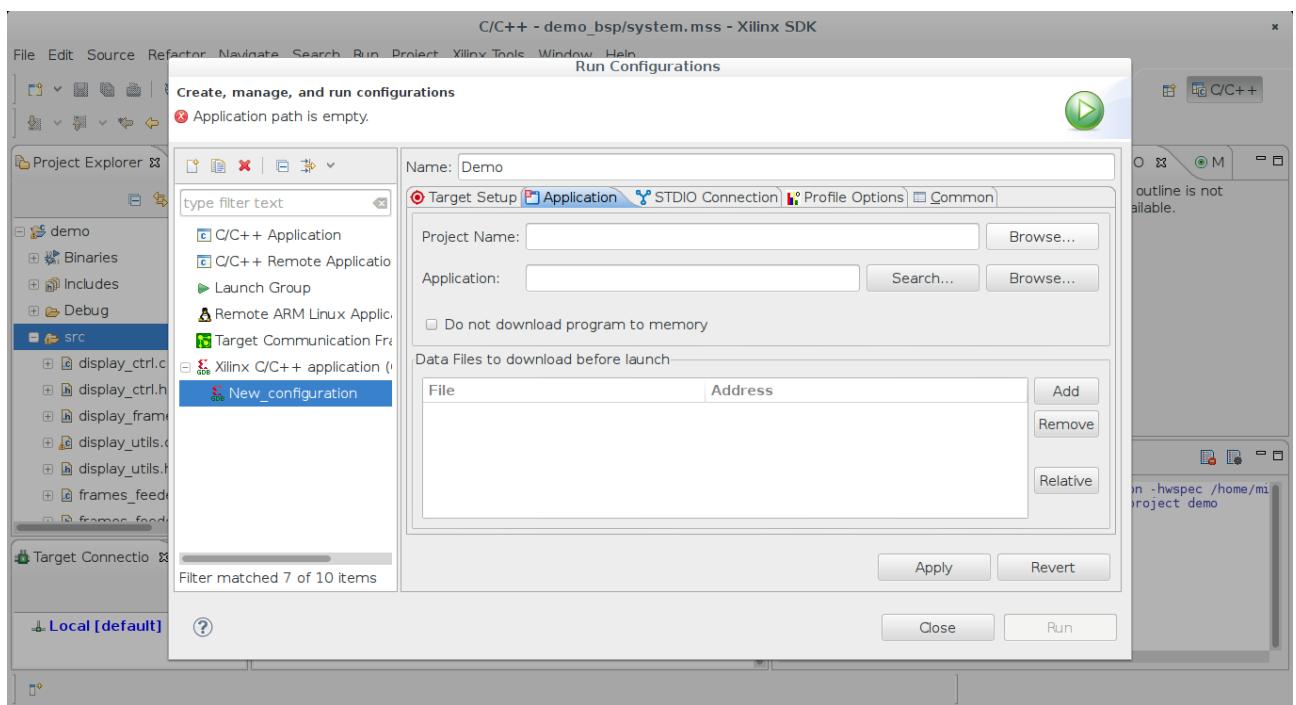
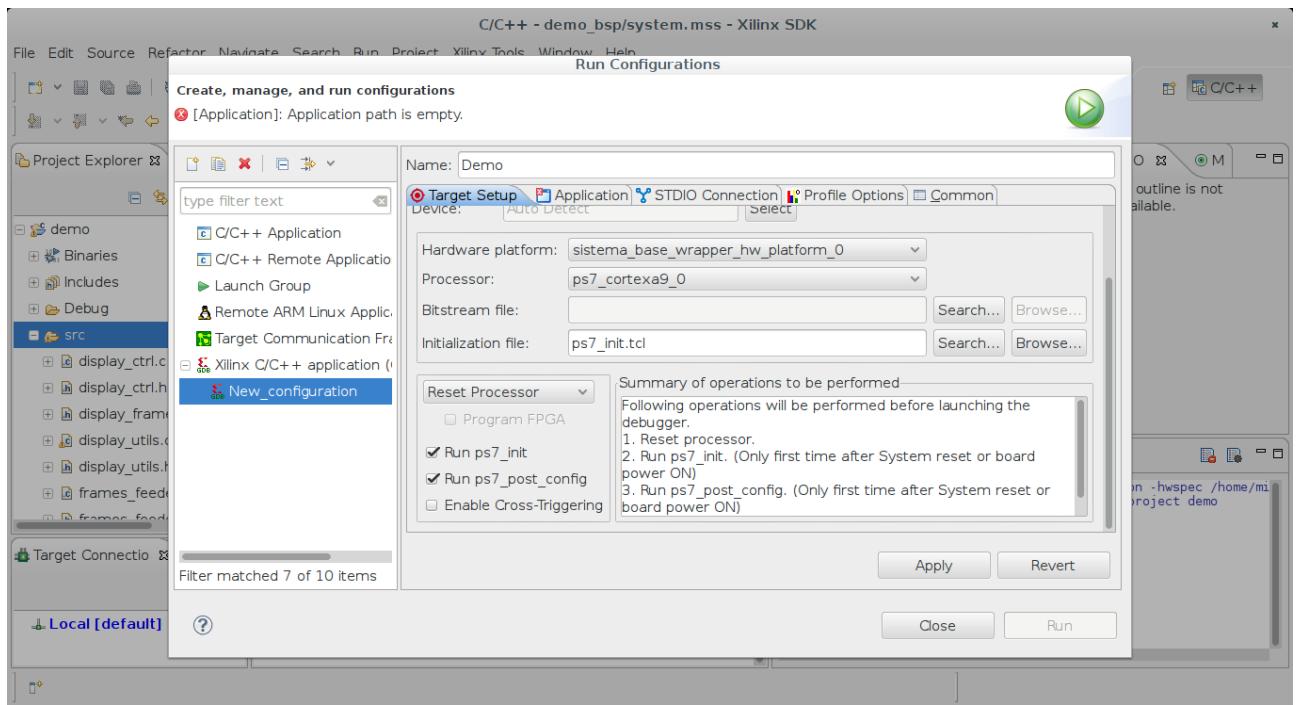
Máscara: 255.0.0.0

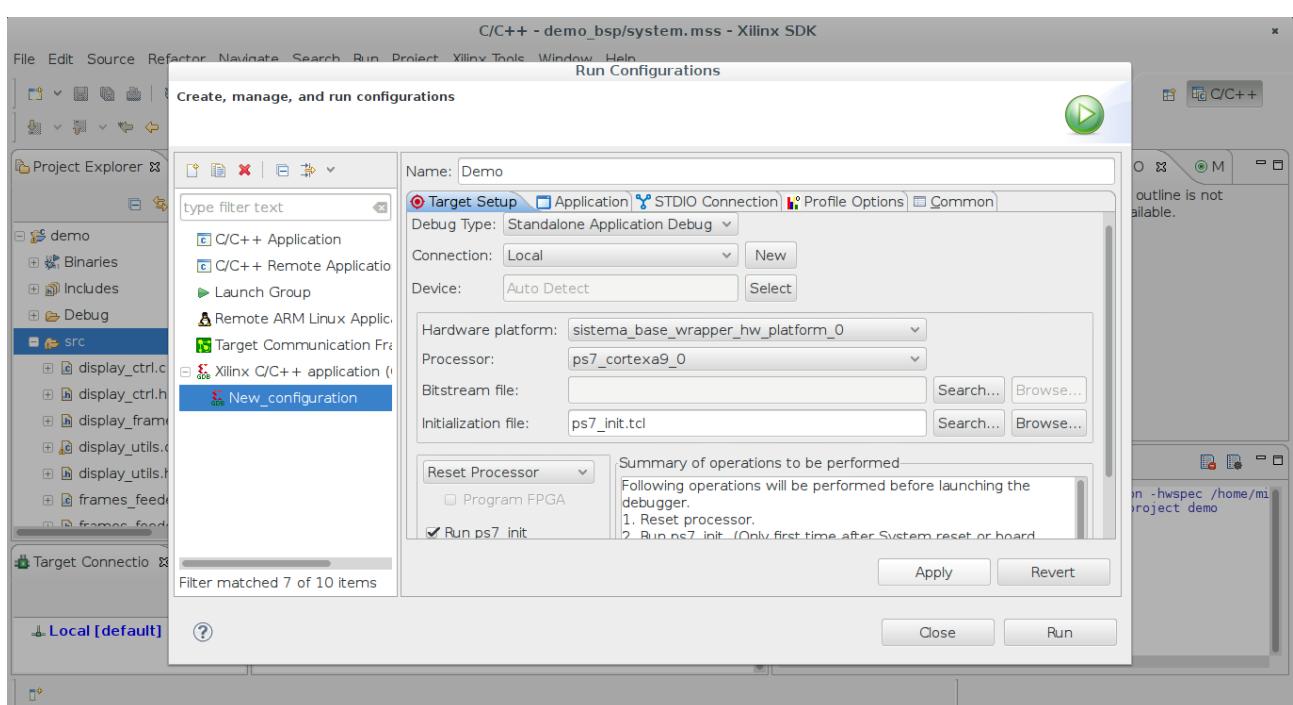
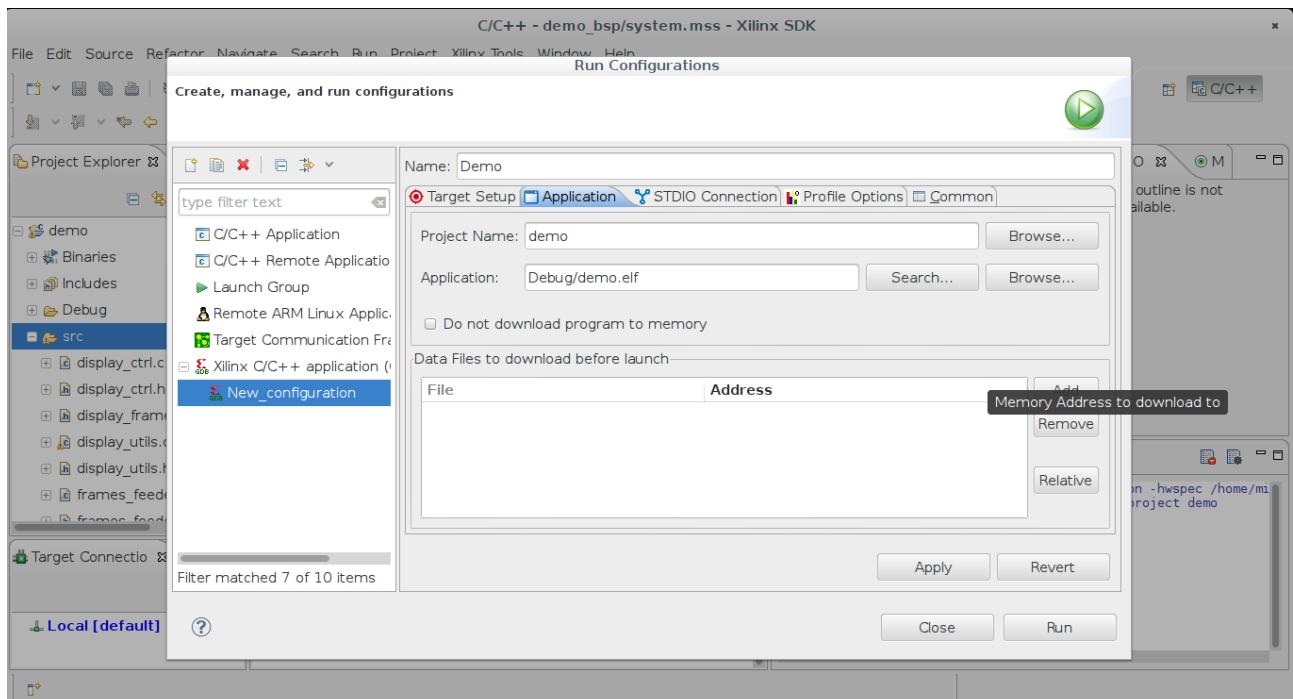
**3.2 – Lanzar servidor en la computadora:** Correr el script computer\_side/server.py

### 3.3 – Lanzar aplicación en Zybo

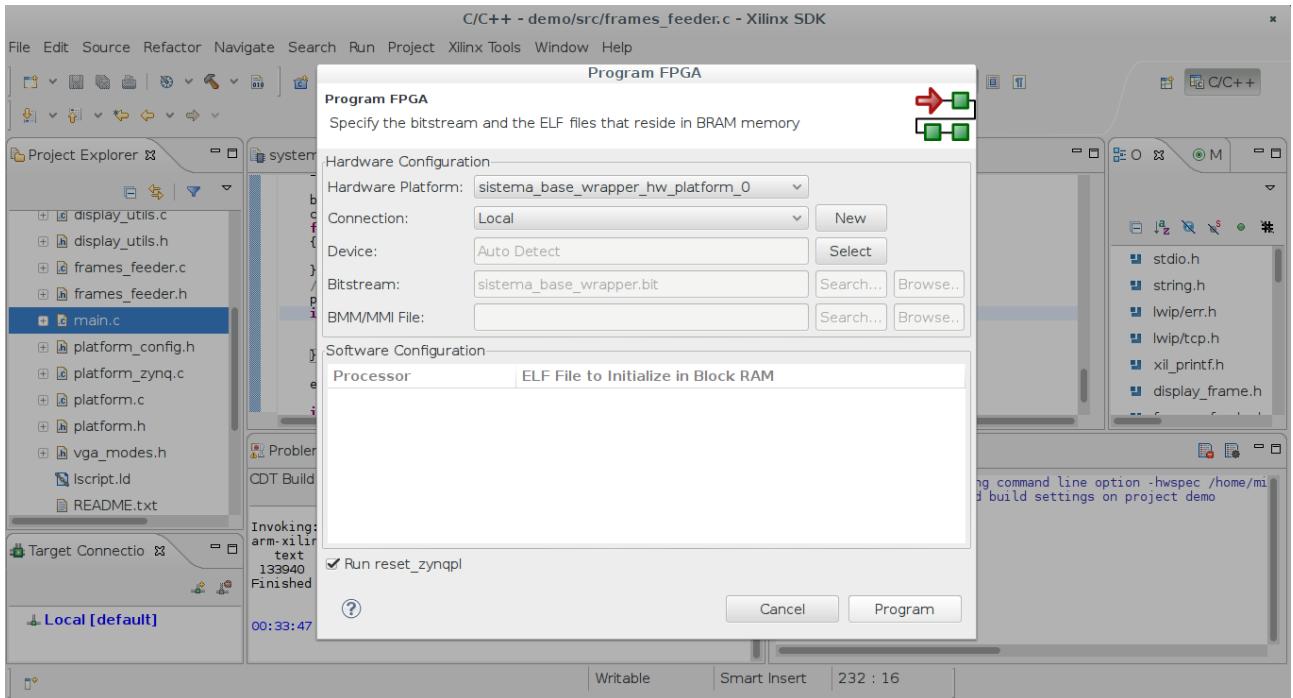
**3.3.1 – Crear “Run Configuration”:** en el menú ir a Run → Run Configurations y clickear el botón “New launch configuration”.



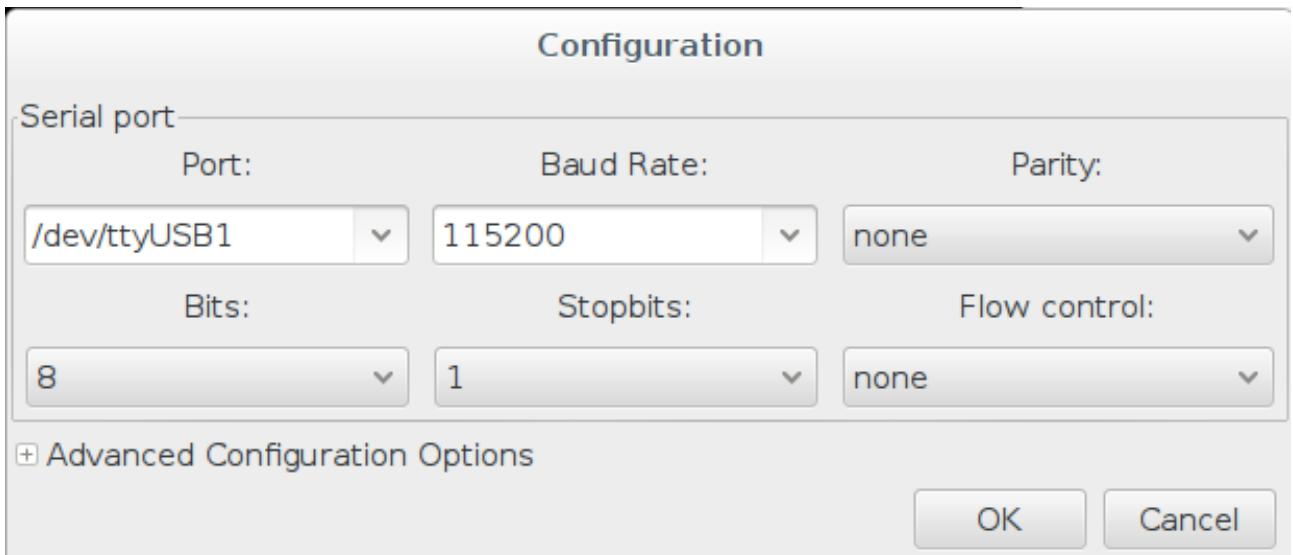




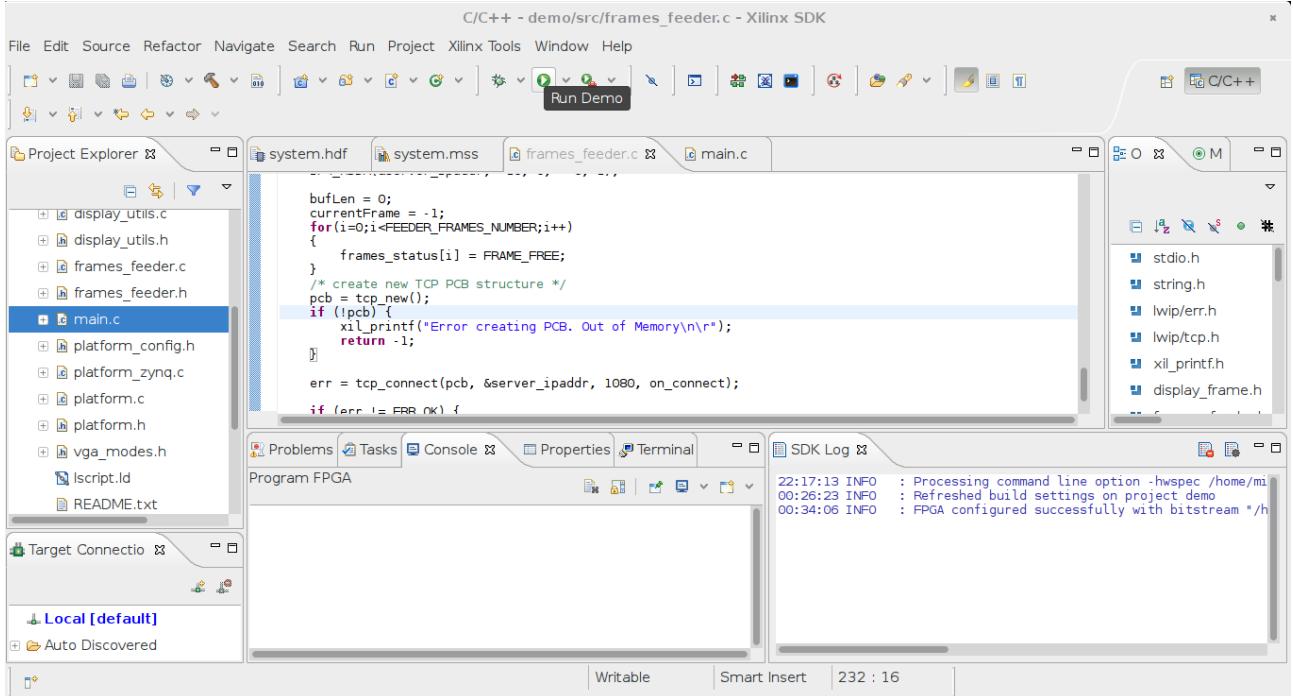
**3.3.2 – Subir bitsream a Zybo, En el menú seleccionar Xilinx Tools → Program FPGA y hacer click en “Program”**



3.3.3 – Conectarse a la UART para ver las salidas del programa, se puede usar cualquier programa para terminales serie, como gtkterm,

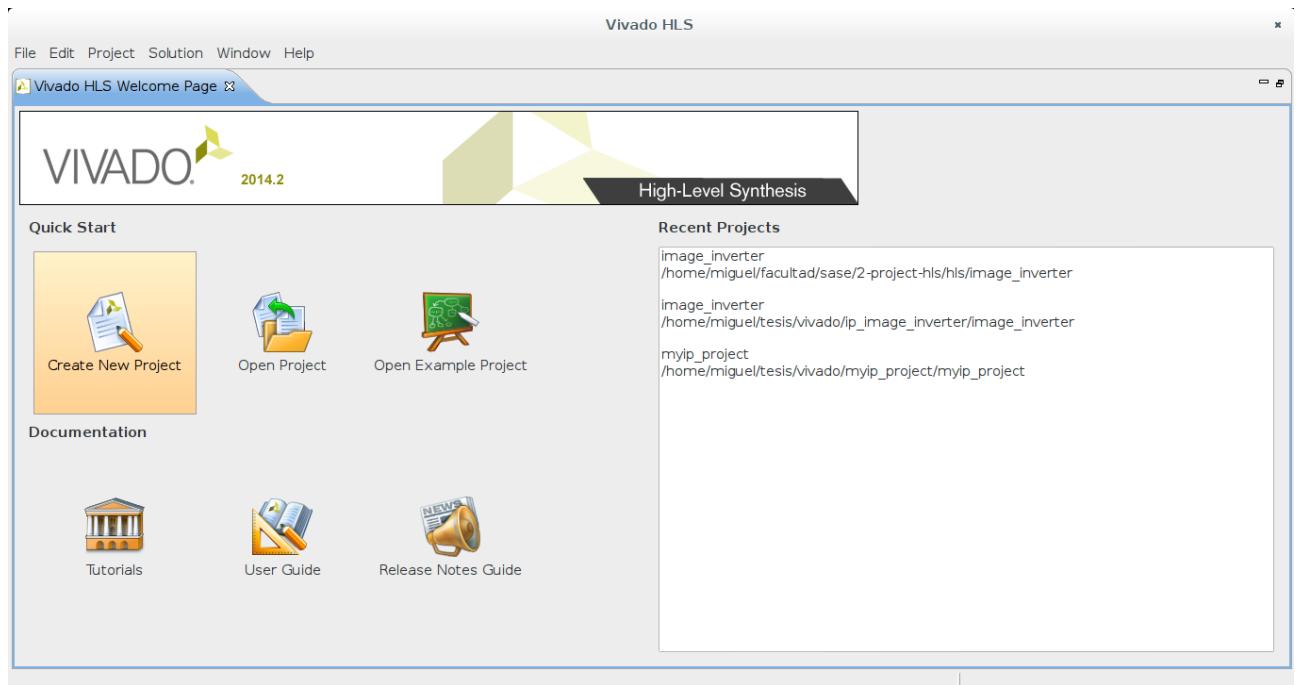


3.3.4 – Lanzar el programa en Zybo haciendo click en el botón verde play

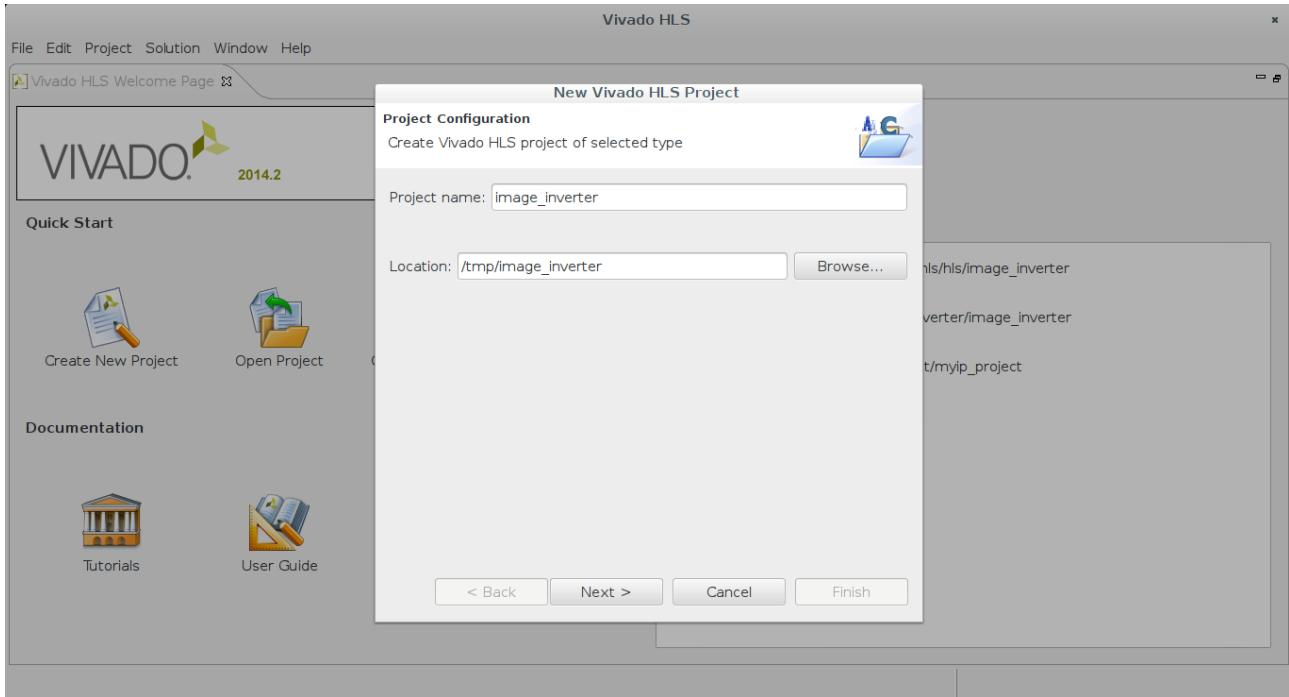


## 4 - Crear proyecto Vivado HLS, correr test en C, sintetizar el diseño y exportar RTL

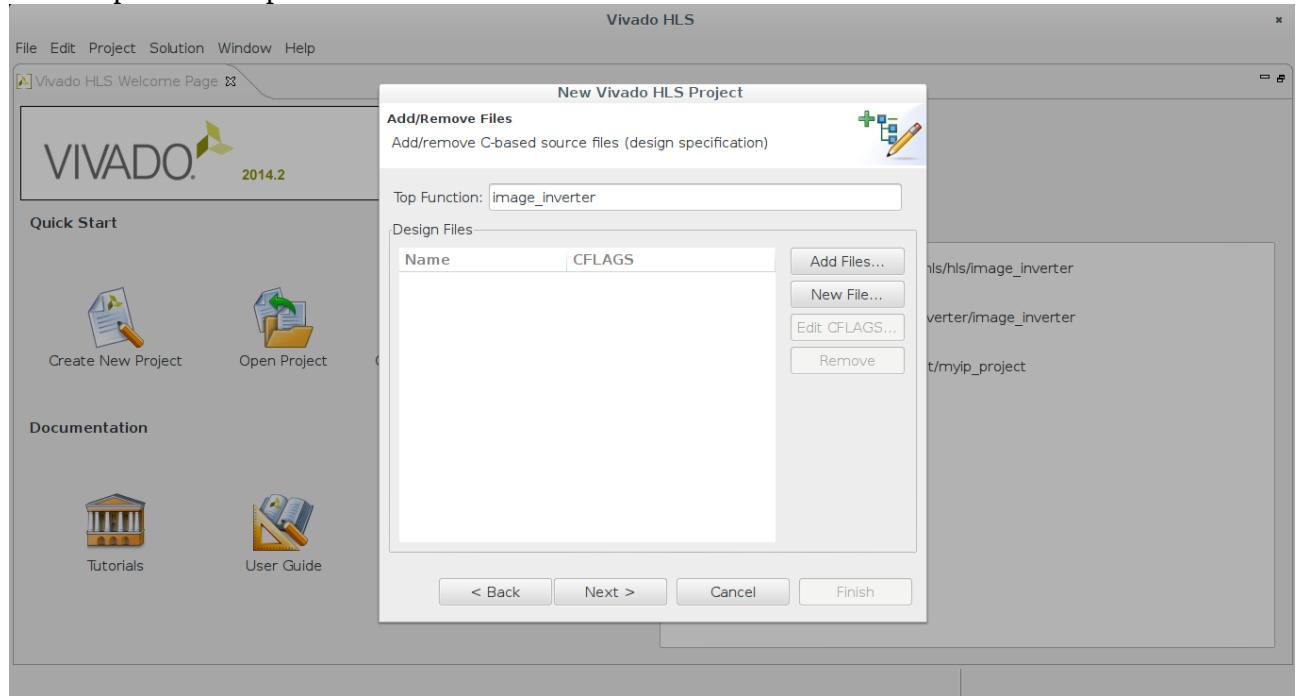
### 4.1 – Crear un nuevo proyecto en Vivado HLS



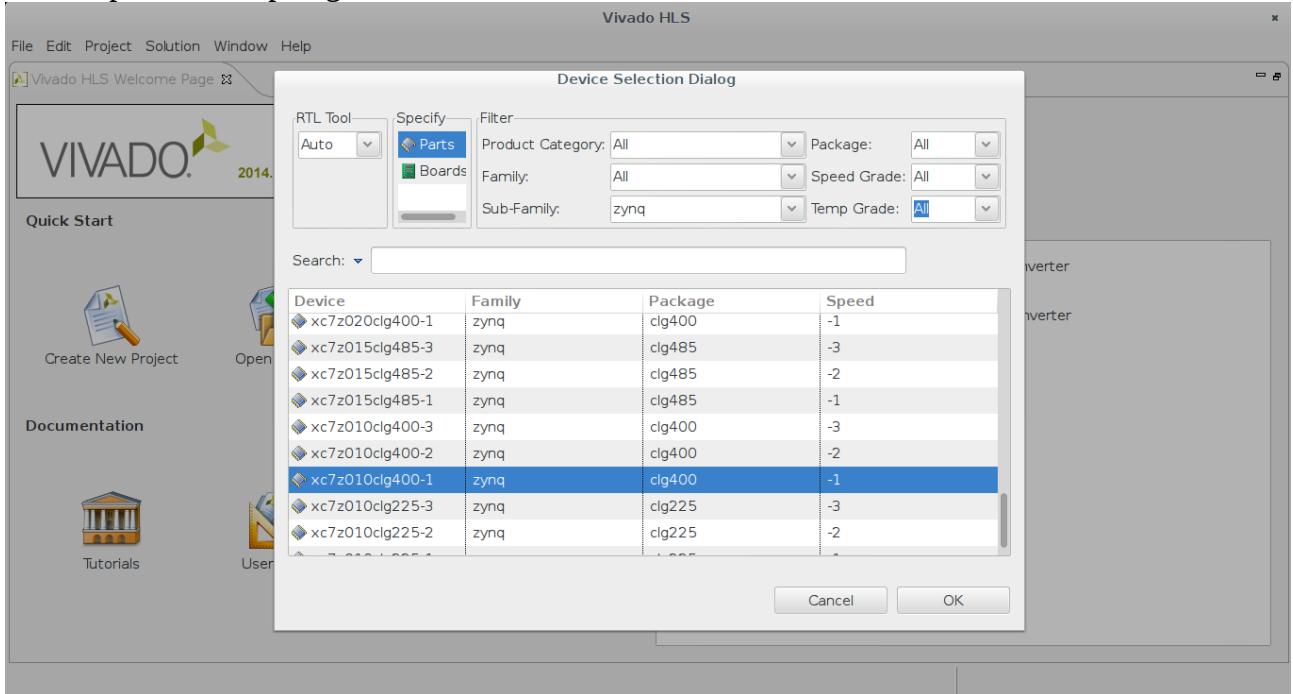
## 4.2 – Especificar nombre y ruta donde guardar el proyecto



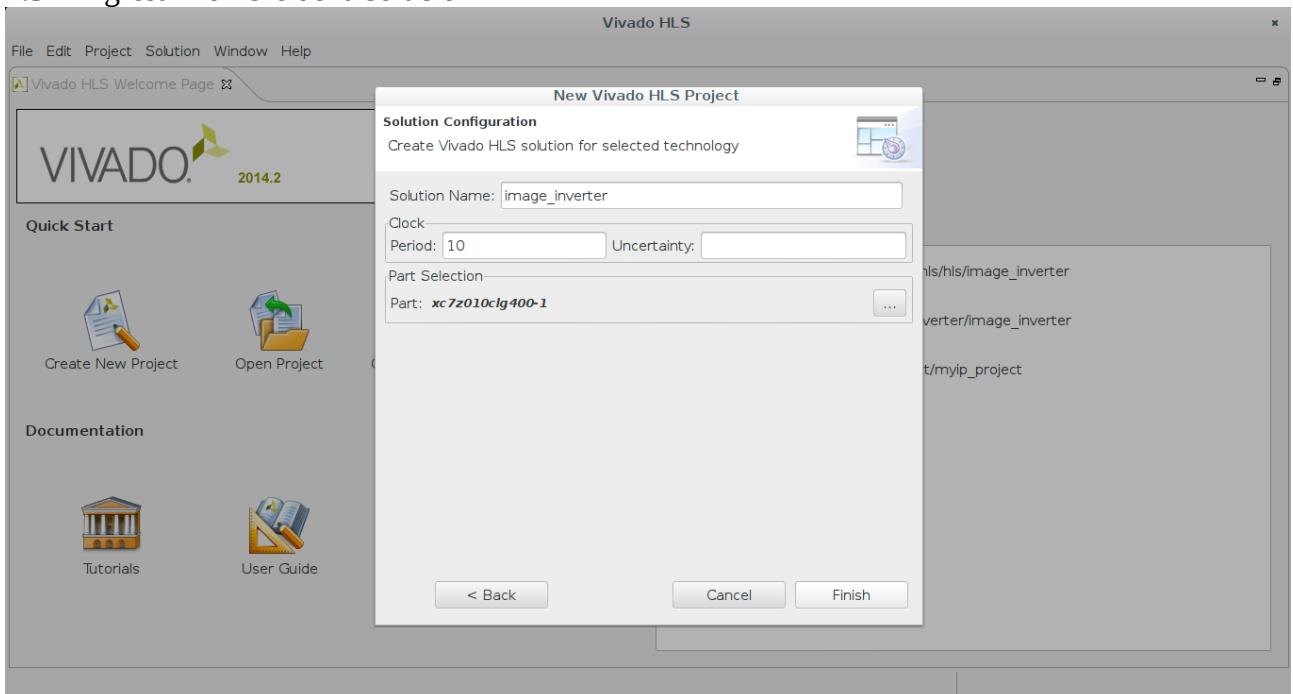
## 4.3 – Especificar Top Function



#### 4.4 – Especificar chip target



#### 4.5 – Ingresar nombre de la solución



#### 4.6 – Crear dentro de Source dos nuevos archivos: image\_inverter.h e image\_inverter.c

Vivado HLS - image\_inverter (/tmp/image\_inverter/image\_inverter)

File Edit Project Solution Window Help

Debug Synthesis Analysis

Explorer image\_inverter image\_inverter.c image\_inverter.h

```
1 ifndef IMAGE_INVERTER_H_
2 define IMAGE_INVERTER_H_
3
4 void image_inverter(volatile int mem[1024*1024*1024*4], unsigned int ptrInput,
5     unsigned int ptrOutput, unsigned int pixelsN);
6
7 endif
8
```

Outline Directive

```
# IMAGE_INVERTER_H_
image_inverter(volatile int[], uns
```

Console Errors Warnings

Writable Smart Insert | 5 : 9 |

This screenshot shows the Vivado HLS IDE interface. The project is named 'image\_inverter'. In the center, there are two tabs: 'image\_inverter.c' and 'image\_inverter.h'. The 'image\_inverter.h' tab is active, displaying a single-line header guard. The 'image\_inverter.c' tab shows a blank implementation block. On the left, the 'Explorer' view shows the project structure with 'image\_inverter' expanded, revealing 'Includes' and 'Source' folders, and 'image\_inverter.c' and 'image\_inverter.h' files under 'Source'. On the right, the 'Outline' and 'Directive' panes are visible, showing the header file's contents. The bottom of the interface features a toolbar with various icons and status bars indicating 'Writable' and line numbers.

Vivado HLS - image\_inverter (/tmp/image\_inverter/image\_inverter)

File Edit Project Solution Window Help

Debug Synthesis Analysis

Explorer image\_inverter image\_inverter.c image\_inverter.h

```
1 #include "image_inverter.h"
2 #include <stdio.h>
3 #include <string.h>
4
5 // BUFFER in Words
6 #define BUF_SIZE 16
7
8 void image_inverter(volatile int mem[1024*1024*1024*4], unsigned int ptrInput, unsigned int
9 {
10    int i, j;
11    unsigned int wordInput = ptrInput / 4;
12    unsigned int wordOutput = ptrOutput / 4;
13    int buf[BUF_SIZE];
14    for(i=0; i < (int) pixelsN-BUF_SIZE; i+=BUF_SIZE)
15    {
16        memcpy(buf, (const void *) &mem[wordInput+i], BUF_SIZE * sizeof(int));
17        for(j=0;j<BUF_SIZE;j++)
18        {
19            buf[j] = ~buf[j];
20        }
21        memcpy((void *) &mem[wordOutput+i], buf, BUF_SIZE * sizeof(int));
22    }
23    for(; i < pixelsN; i++)
24    {
25        mem[wordOutput+i] = ~mem[wordInput+i];
26    }
27
28}
```

Outline Directive

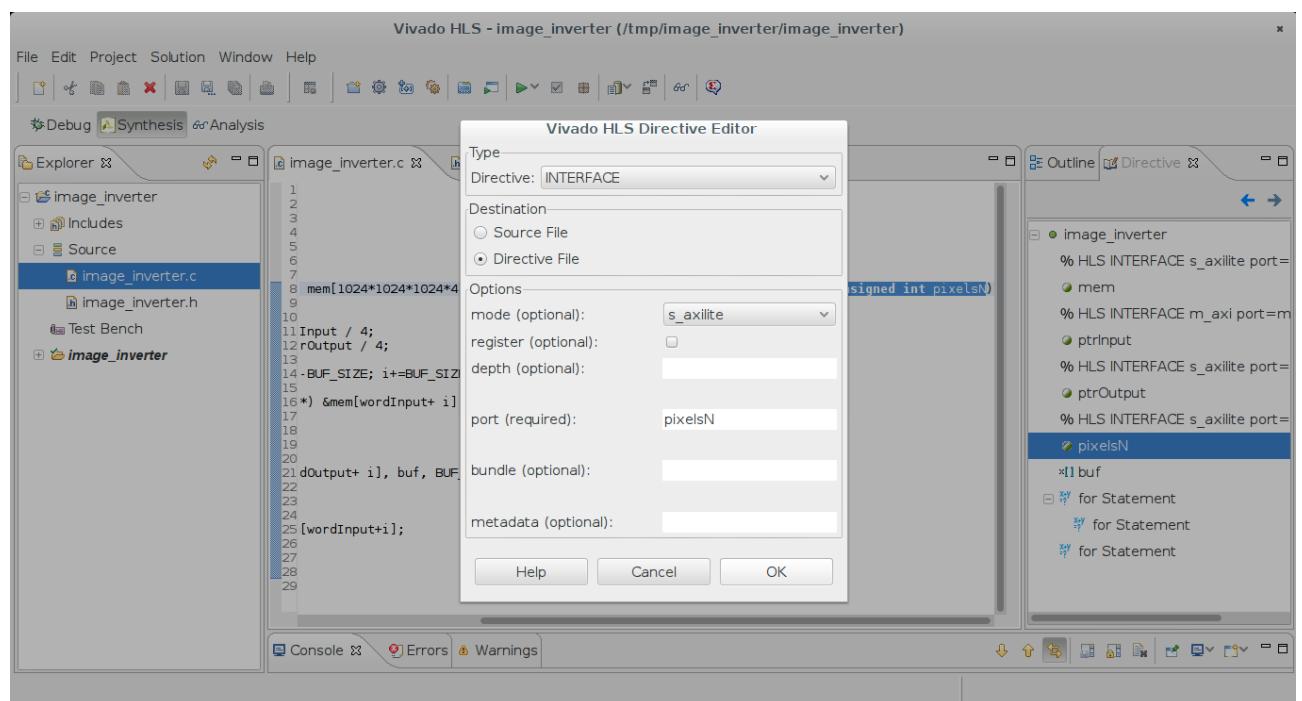
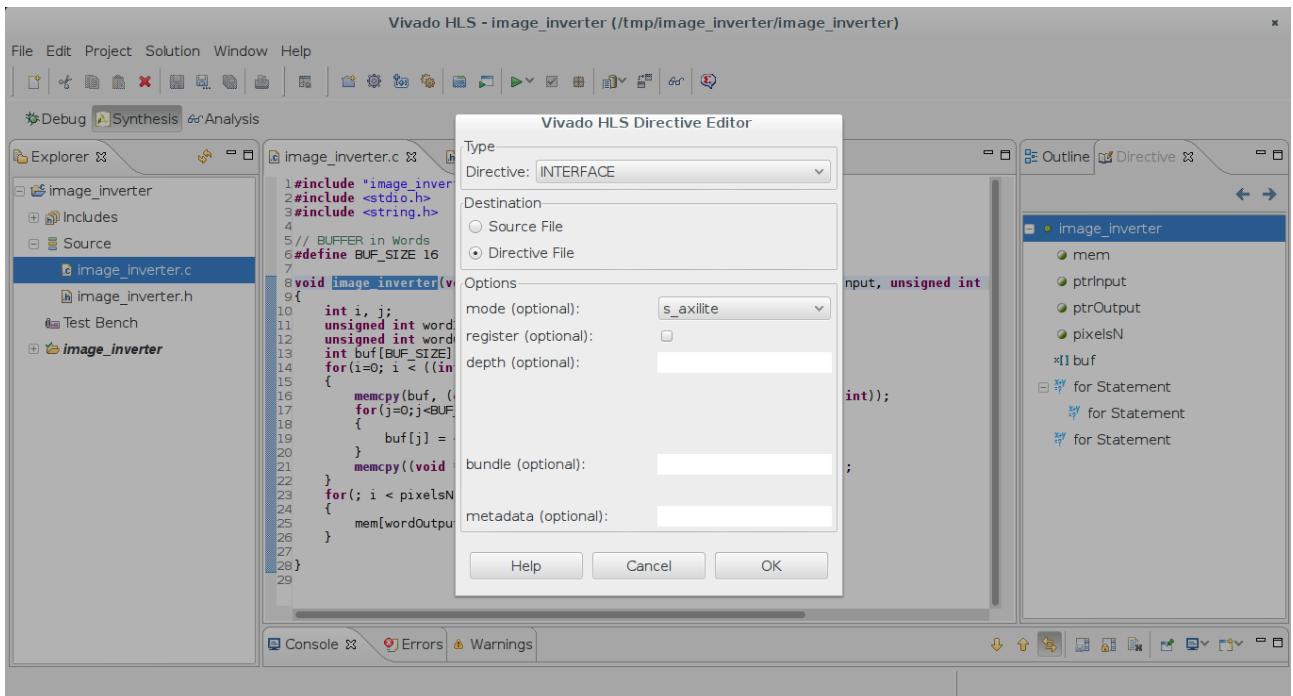
```
image_inverter.h
stdio.h
string.h
# BUF_SIZE
image_inverter(volatile int[], uns
```

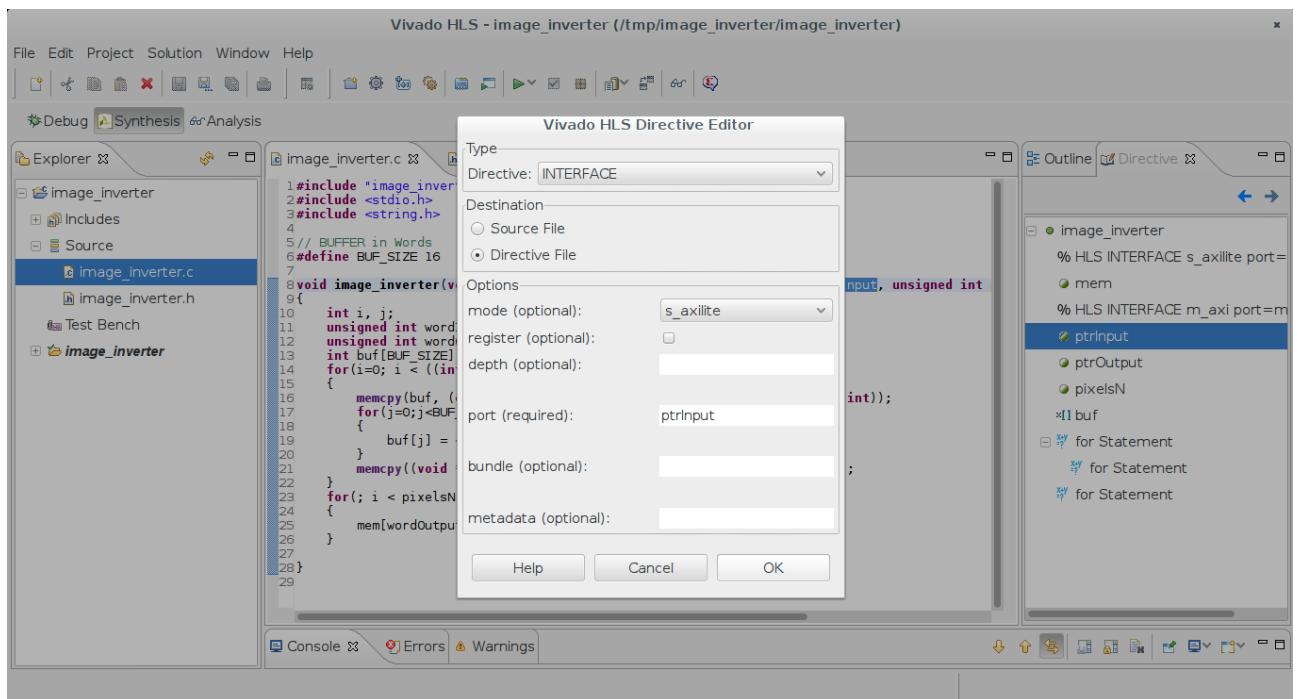
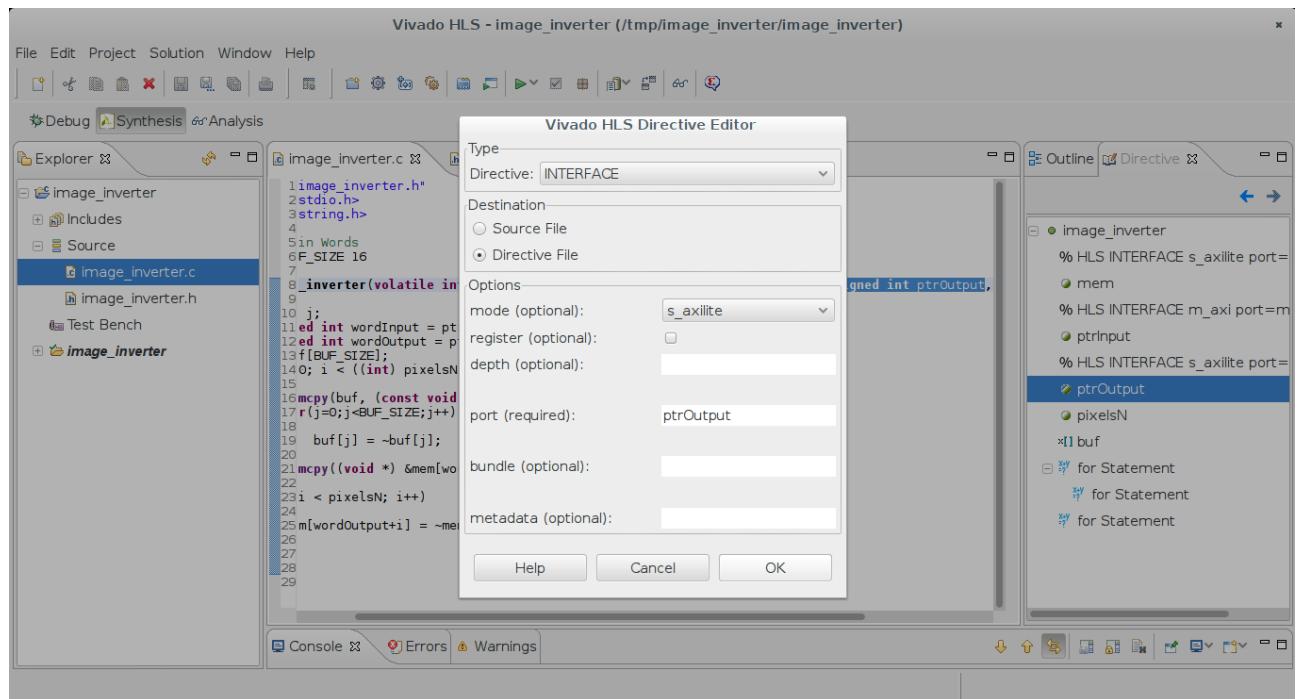
Console Errors Warnings

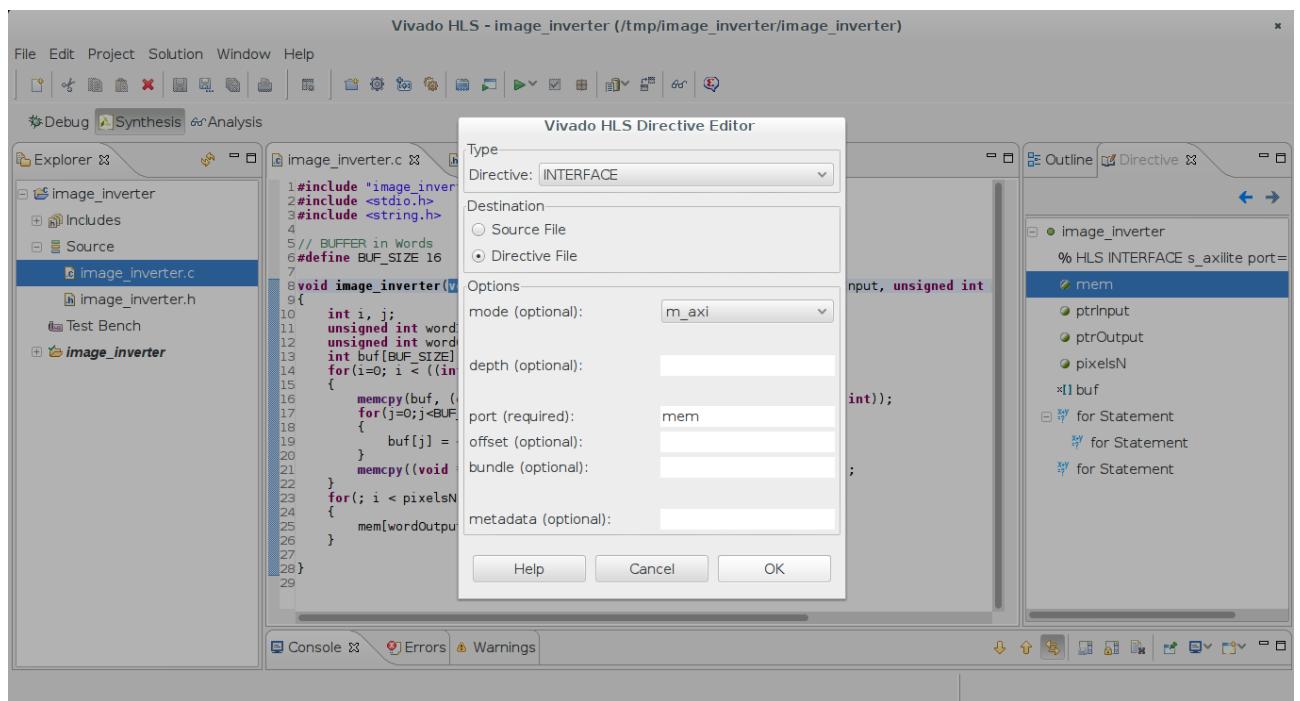
Writable Smart Insert | 26 : 6 |

This screenshot shows the Vivado HLS IDE after modifications. The 'image\_inverter.c' tab is now active, displaying a C function implementation. The code uses 'memcpy' to copy word-sized blocks from memory, inverts each word in the block, and then copies it back. It also handles the remaining words in the image. The 'image\_inverter.h' tab remains the same as in the previous screenshot. The 'Outline' and 'Directive' panes are updated to reflect the changes in the implementation. The bottom of the interface shows the status bar with 'Writable' and line number 26:6.

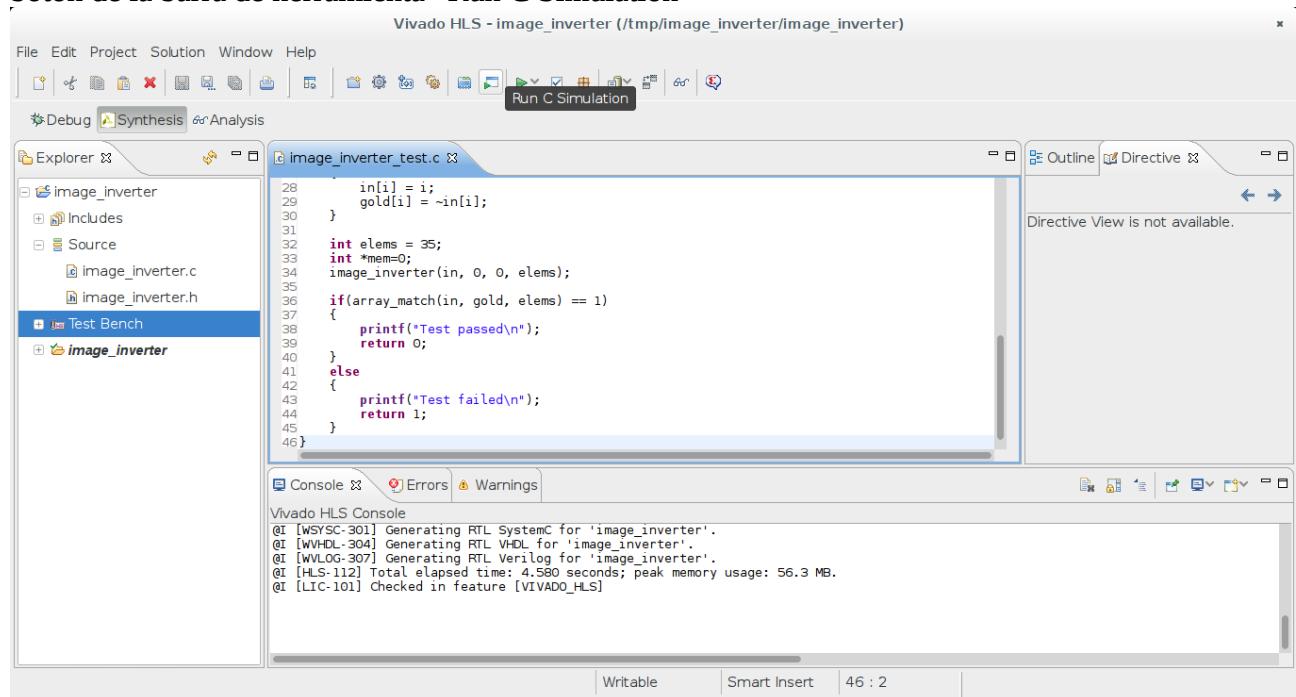
## 4.7 – Especificar las directivas que definen la interfaz del IP







**4.8 – Test en C:** Crear un nuevo archivo `image_inverter_test.c` en Test Bench, correr el test con el botón de la barra de herramientas “Run C Simulation”



## 4.9 – Sintetizar IP y ver resultados de consumo de recursos y performance

Vivado HLS - image\_inverter (/tmp/image\_inverter/image\_inverter)

File Edit Project Solution Window Help

Debug Synthesis Analysis

Run C Synthesis

Explorer image\_inverter image\_inverter.c image\_inverter.h

```

1 #include "image_inverter.h"
2 #include <stdio.h>
3 #include <string.h>
4
5 // BUFFER in Words
6 #define BUF_SIZE 16
7
8 void image_inverter(volatile int mem[1024*1024*1024*4], unsigned int ptrInput, unsigned int
9 {
10     int i, j;
11     unsigned int wordInput = ptrInput / 4;
12     unsigned int wordOutput = ptrOutput / 4;
13     int buf[BUF_SIZE];
14     for(i=0; i < ((int) pixelsN)-BUF_SIZE; i+=BUF_SIZE)
15     {
16         memcpy(buf, (const void *) &mem[wordInput+ i], BUF_SIZE * sizeof(int));
17         for(j=0;j<BUF_SIZE;j++)
18         {
19             buf[j] = ~buf[j];
20         }
21         memcpy((void *) &mem[wordOutput+ i], buf, BUF_SIZE * sizeof(int));
22     }
23     for( i < pixelsN; i++)
24     {
25         mem[wordOutput+i] = ~mem[wordInput+i];
26     }
27 }
28
29

```

Console Errors Warnings

Writable Smart Insert 17 : 32

Vivado HLS - image\_inverter (/tmp/image\_inverter/image\_inverter)

File Edit Project Solution Window Help

Debug Synthesis Analysis

Explorer image\_inverter image\_inverter.c image\_inverter.h image\_inverter\_csynth.rpt

General Information

- Date: Tue Aug 11 20:31:50 2015
- Version: 2014.2 (Build 932637 on Wed Jun 11 12:38:34 PM 2014)
- Project: image\_inverter
- Solution: image\_inverter
- Product family: zynq zynq\_fpv6
- Target device: xc7z010clg400-1

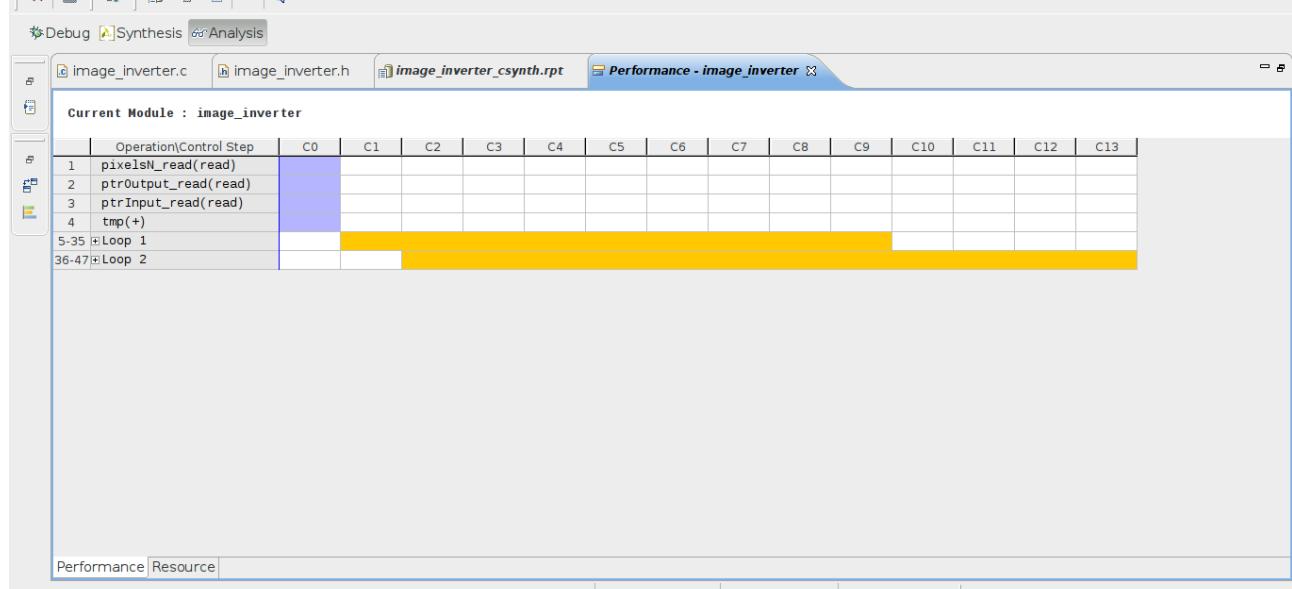
Performance Estimates

Timing (ns)

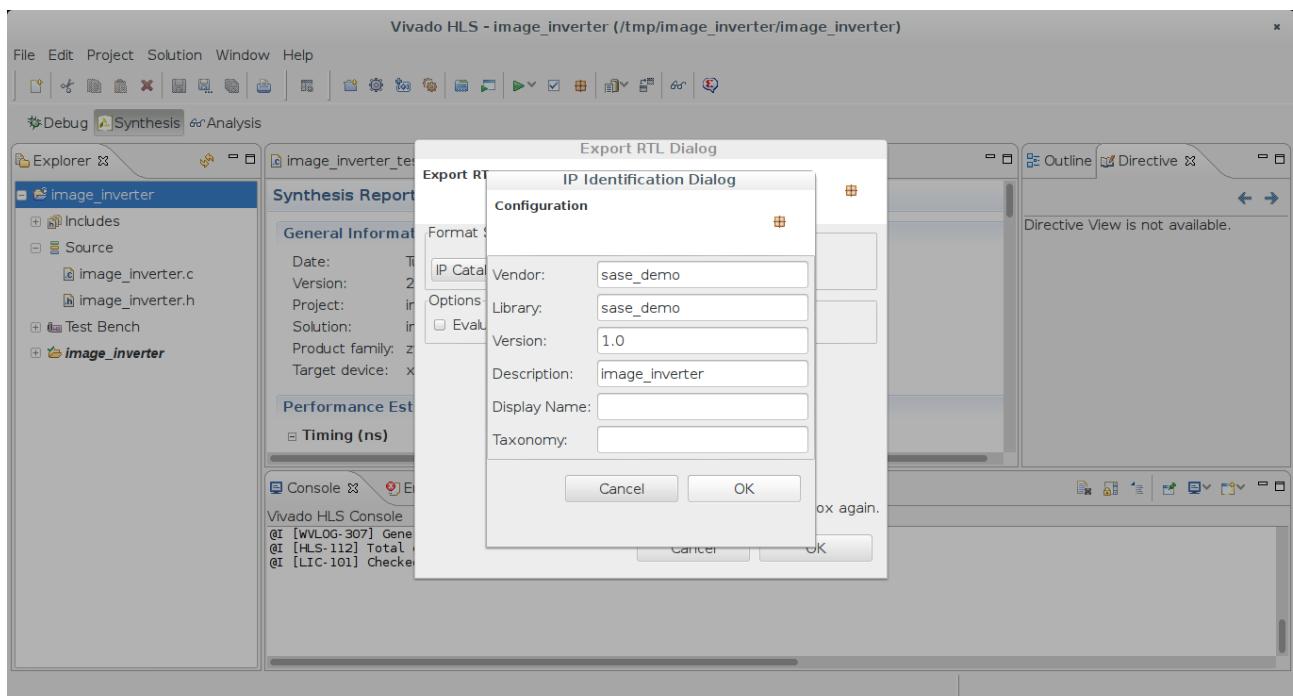
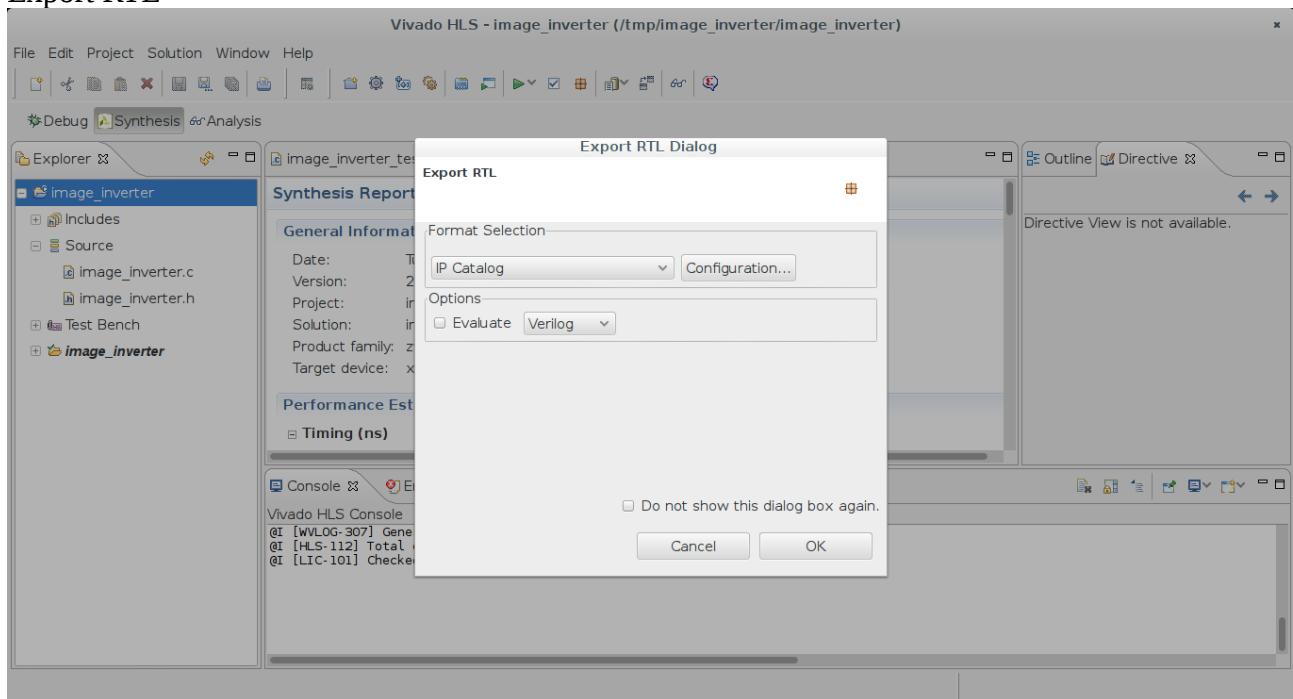
Summary

Clock Target Estimated Uncertainty

Console Errors Warnings

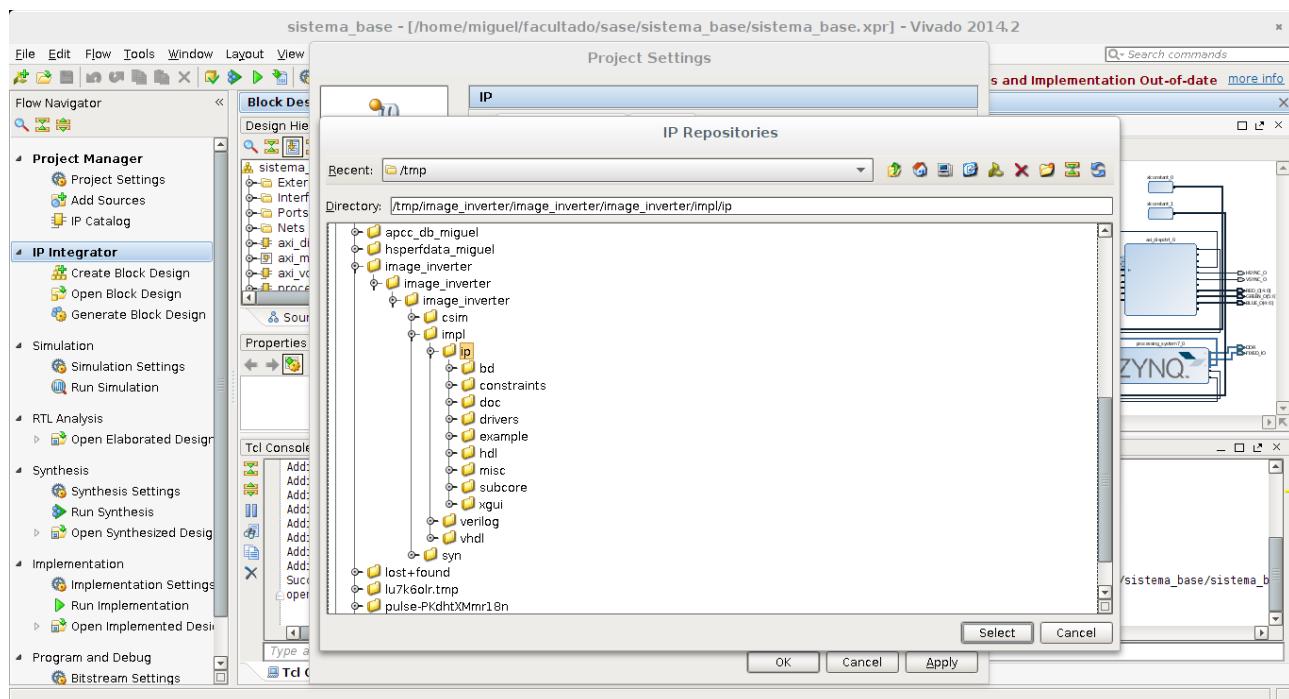
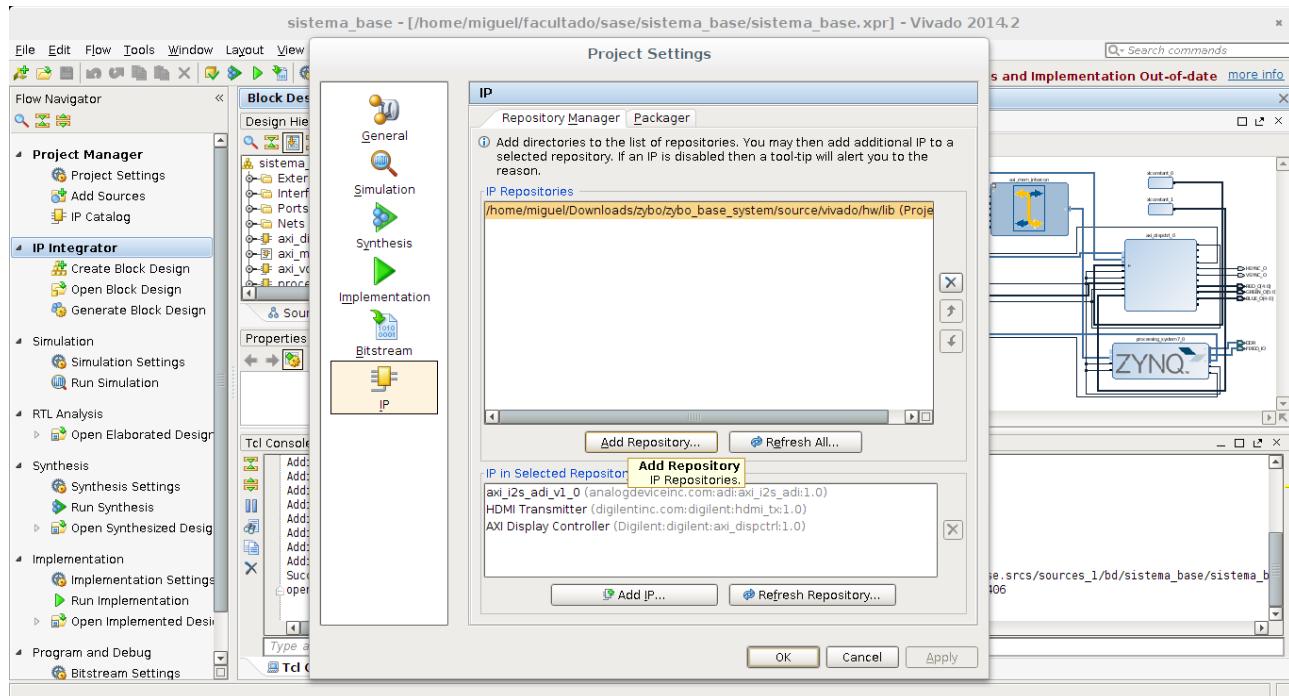


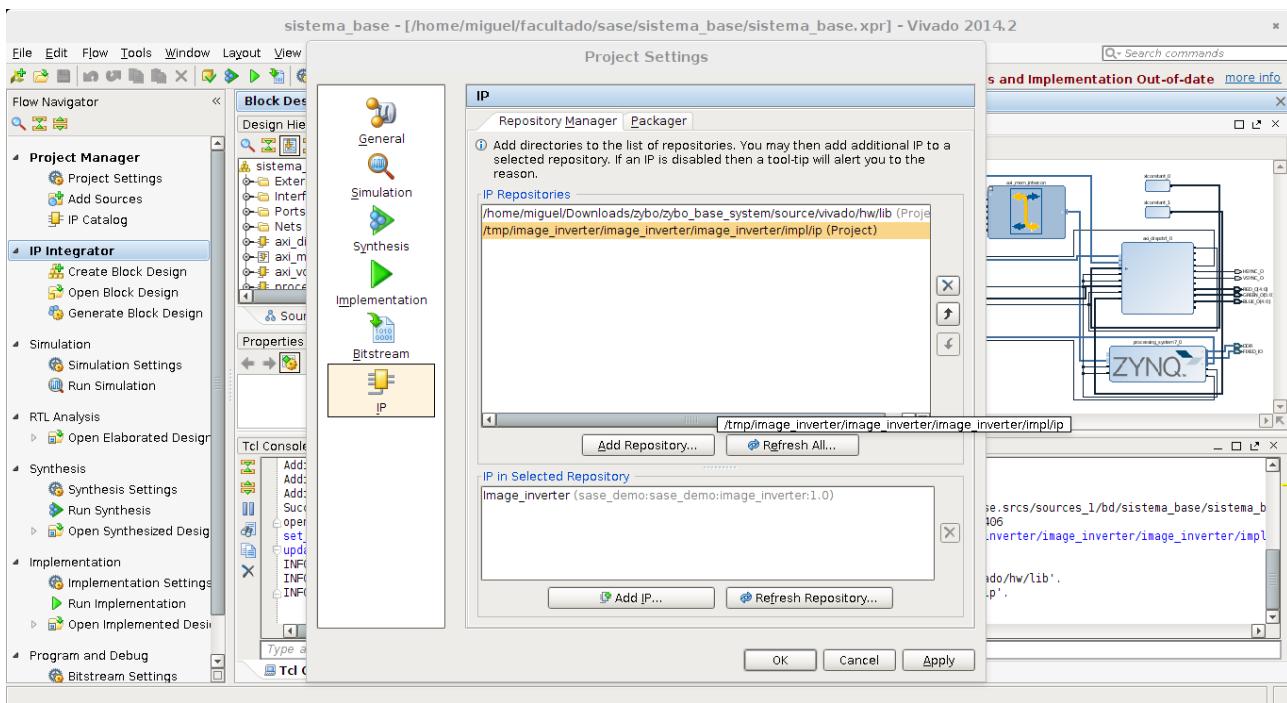
#### 4.10 – Exportar RTL para poder utilizar el IP creado en Vivado IP Integrator: Ir al menú Solution → Export RTL



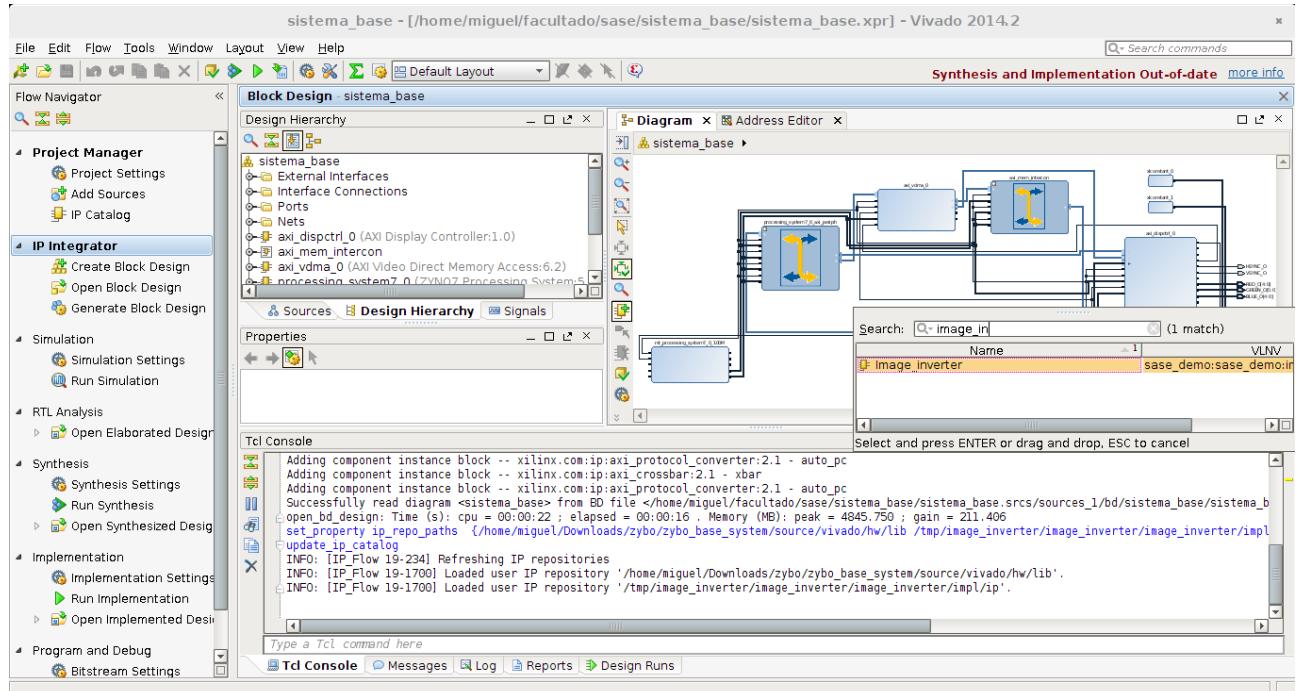
## 5 – Importar IP en Vivado e Integrar IP al diseño

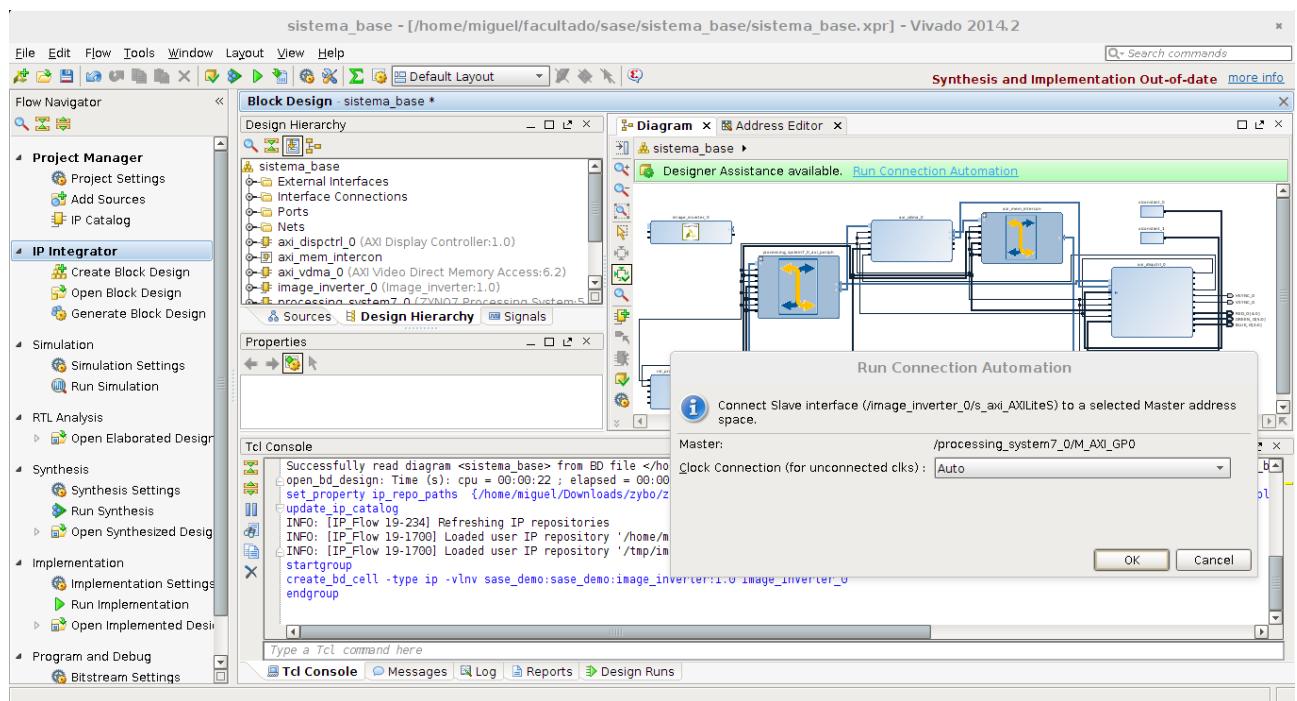
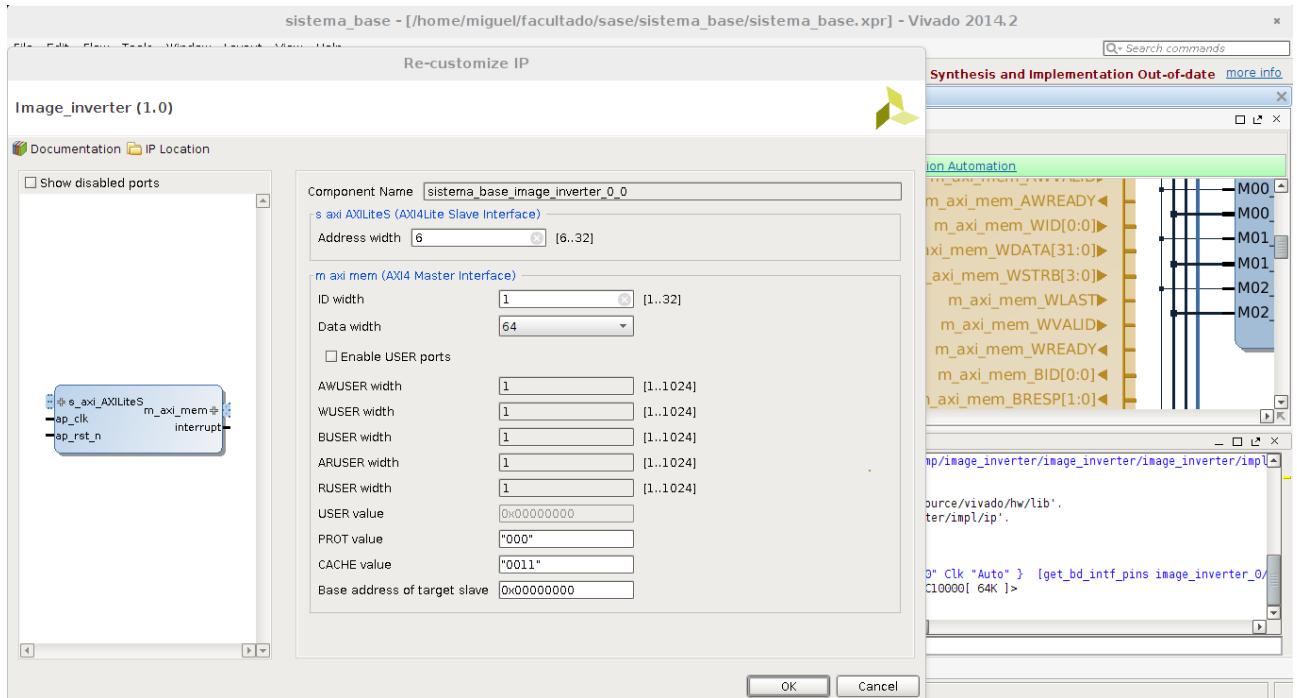
5.1 – Importar el IP Image Inverter: En el menú ir a Tools → Project Settings, en la ventana que se abre seleccionar “IP”. Hacer click en “Add Repository” e introducir la ruta del IP exportado desde Vivado HLS. Hacer click en “Select” y luego en “Ok”.

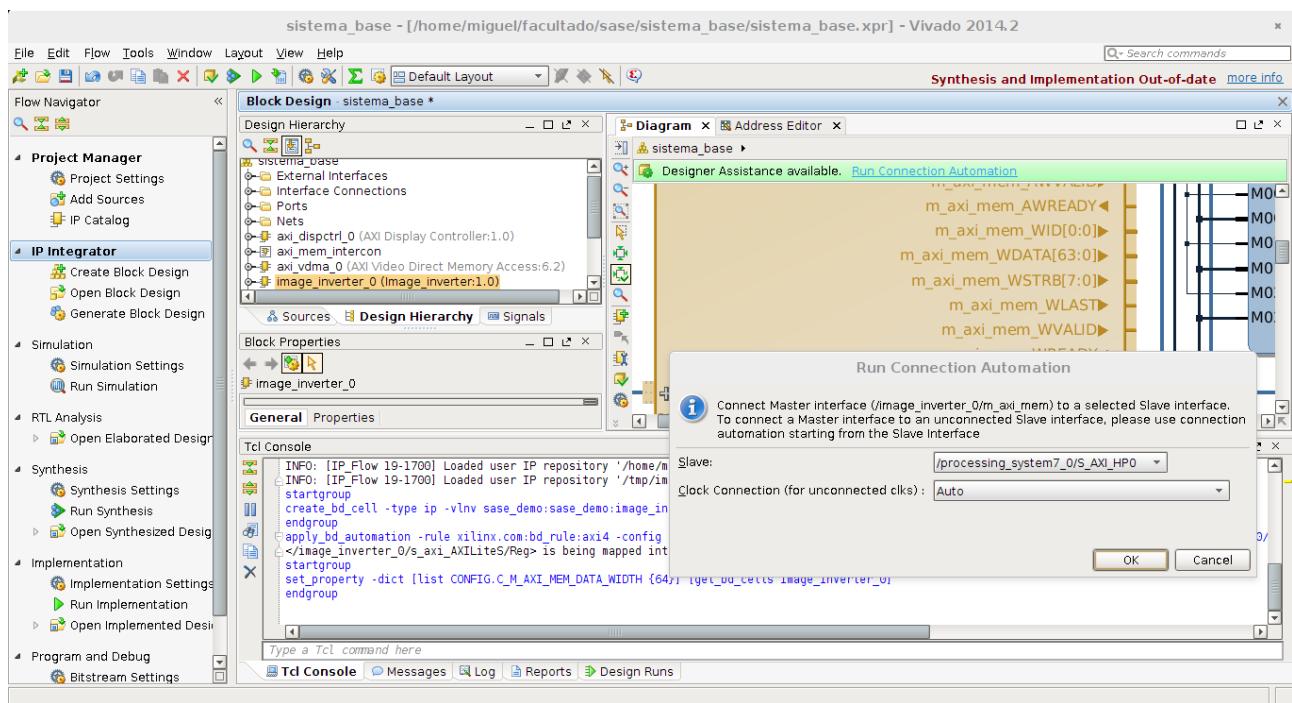




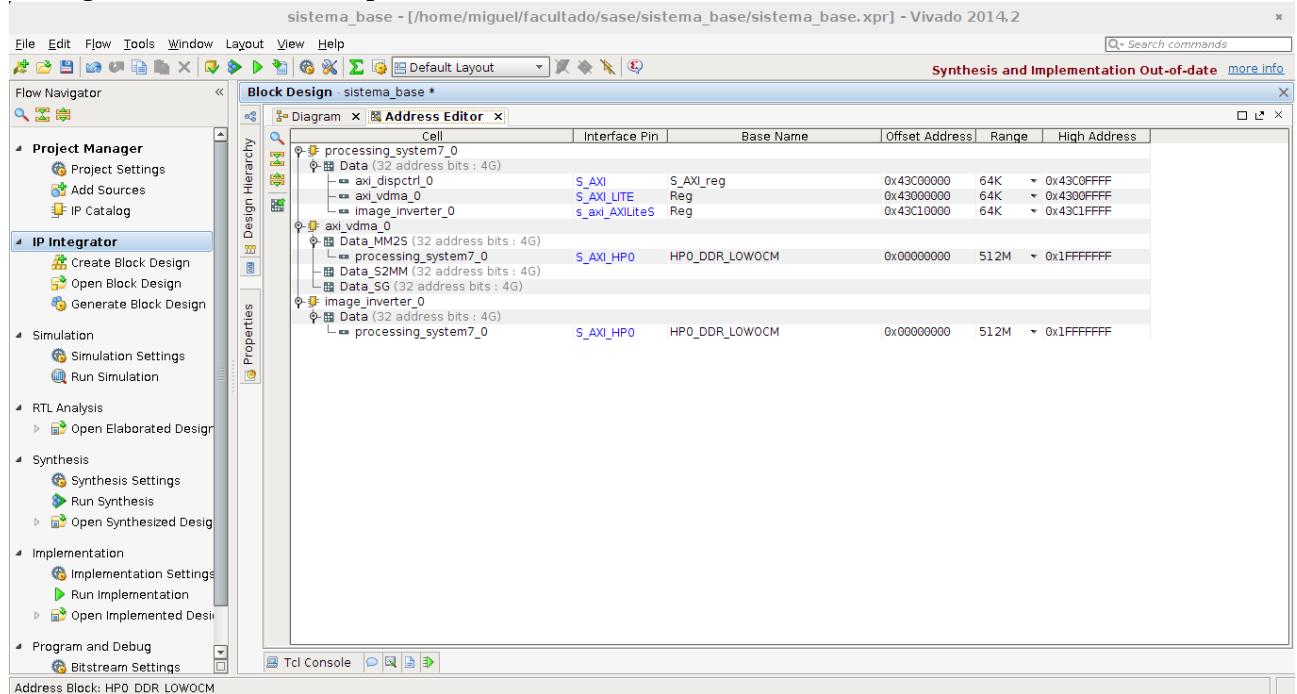
**5.2 – Agregar al Block Design el IP Image Inverter, configurarlo a través de la opción “Customize Block” del menú contextual y conectarlo al resto del circuito mediante la funcionalidad “Run Connection Automation”**







### 5.3 – Configurar espacio de direcciones del IP agregado, mediante la opción del menú contextual “Assign Address” en la solapa “Address Editor”



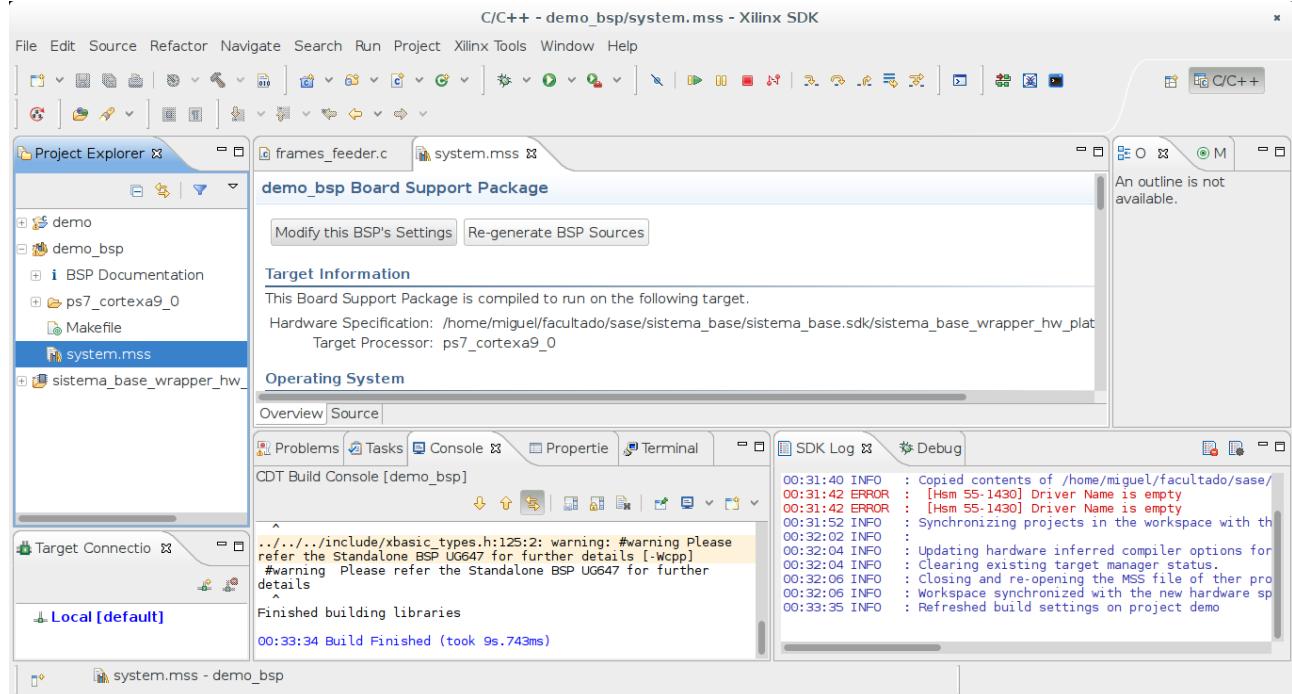
### 5.4 – Generar bitstream haciendo click en “Generate Bitstream”, en el menú Flow Navigator.

## 6 - Exportar a SDK, modificar aplicación para usar el nuevo IP

6.1 – En Vivado IP Integrator seleccionar la opción de menú File → Export → Export Hardware y hacer click en “Ok” en el dialogo emergente.

**6.2** – Lanzar Vivado SDK (si no estaba abierto), En Vivado IP Integrator seleccionar la opción de menú File → Launch SDK, hacer click en “Ok” en el dialogo emergente.

**6.3** – Re generar fuentes del BSP para actualizar drivers, hacer click en el botón “Re-generate BSP Sources”



**6.4** – Modificar el código de la aplicación para que haga uso del nuevo IP