Cómo aumentar los pines de una Zynq a través de una FPGA expansora

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/

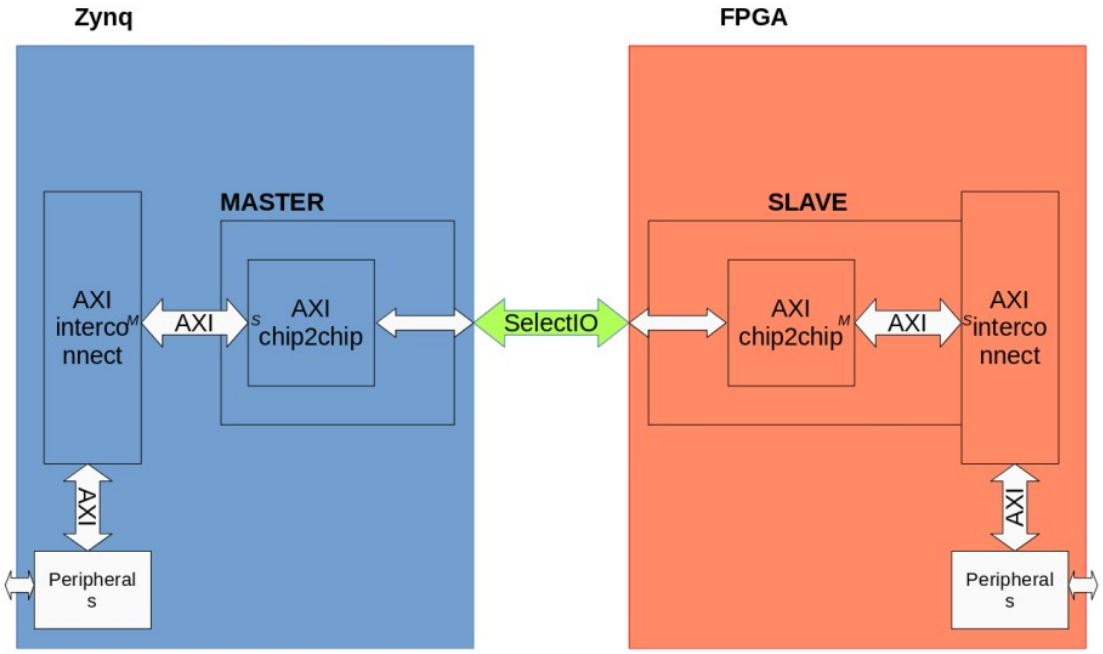
Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

Una situación poco común, pero que puede existir, es querer utilizar una Zynq que tiene menos pines de los que se necesitan para realizar una tarea. Bien, pues existe una forma de aumentar el número de pines de la Zynq utilizando una FPGA a modo de expansora y prolongando el bus AXI de la Zynq a esta expansora.

Esto se realiza haciendo uso del bloque IP *AXI_chip2chip*, pero existen varias formas de hacerlo (unas haciendo uso de los pines GT de los dispositivos de Xilinx y otras sin hacer uso de este).



Pero el modelo general es el mismo, es el que está plasmado en la imagen anterior.

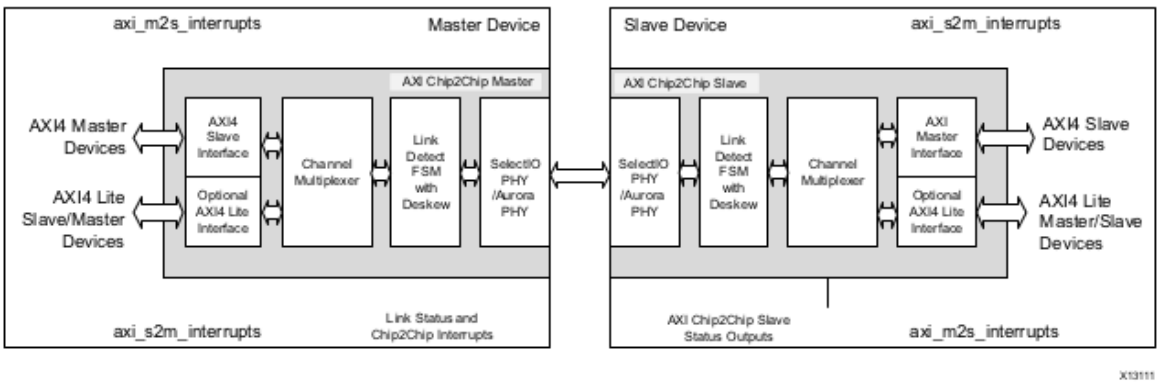


Figure 1-1: **AXI Chip2Chip Block Diagram**

NOTA: Es importante entender que el *AXI_interconnect* de la Zynq lo que va a hacer es mandar una «orden» a una dirección de memoria tanto de la Zynq como de la expansora (a través del *Aurora* o del *SelectIO*) y si hay un dispositivo que tenga adjudicado esa dirección de memoria atenderá a la comunicación. Por eso es importante que las direcciones de memoria de la expansora **NO** colisionen con las de la Zynq, por ello el *AXI_chip2chip* de la **Zynq** tendrá unas direcciones de memoria con un espacio de memoria reservado que ningún dispositivo de la Zynq podrá pisar. Y en la **Expansora**, las direcciones de memoria **comenzarán en la dirección de memoria del *AXI_chip2chip* de la Zynq** y **NO podrán sobrepasar el espacio de memoria reservado** para éste.

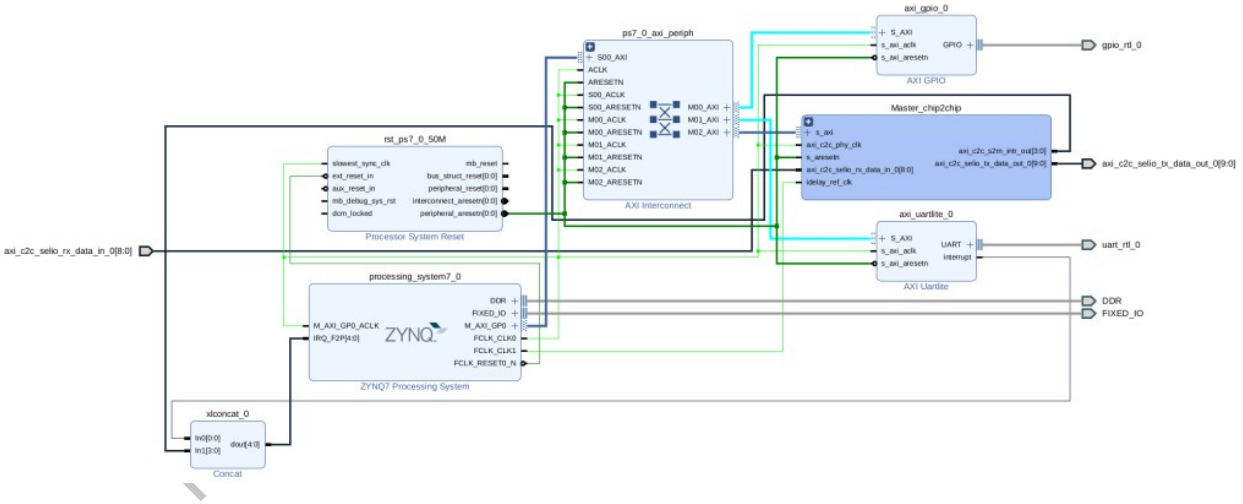
Para facilitar el entendimiento de la configuración vamos proponer dos ejemplos iguales, uno haciendo uso de los pines *GT*, con el *Aurora 8b/10b*, y otro sin usarlos, utilizando el *SelectIO* de serie.

Ejemplo 1

(*Este ejemplo hará uso del bus del sistema de comunicación **SelectIO***)

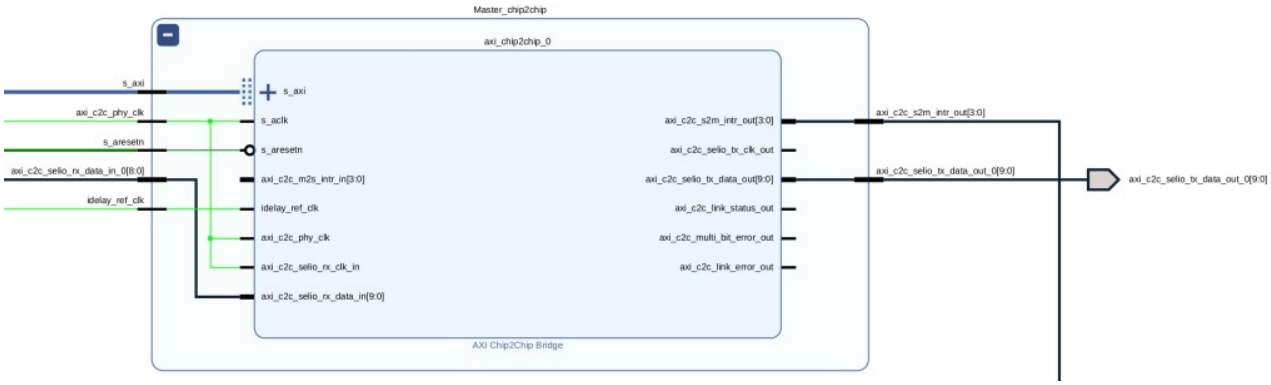
En este ejemplo, tenemos una Zynq se nos ha quedado corta de pines, y necesitamos todavía un bloque AXI_I2C y un AXI_UART, para ello se quiere utilizar una FPGA a modo de expansora.

Configuración de la Zynq



En la imagen anterior se puede ver como están los bloques IP, y además también aparece un bloque acoplado al AXI_interconnect que será el que permita comunicarse con la expansora a través del AXI.

Si nos fijamos en el bloque de comunicación, se puede ver que lo único que hay es un AXI_chip2chip. Este bloque IP realizará la comunicación utilizando el protocolo **SelectIO DDR**. Para ello requiere de puertos de entrada y de salida que se comuniquen con el AXI_chip2chip de la expansora, estos puertos son el **axi_c2c_selio_rx_data_in** y el **axi_c2c_selio_data_out**. Además, también hay unos relojes que acompañan la comunicación. El resto de puertos son de salida son de control.



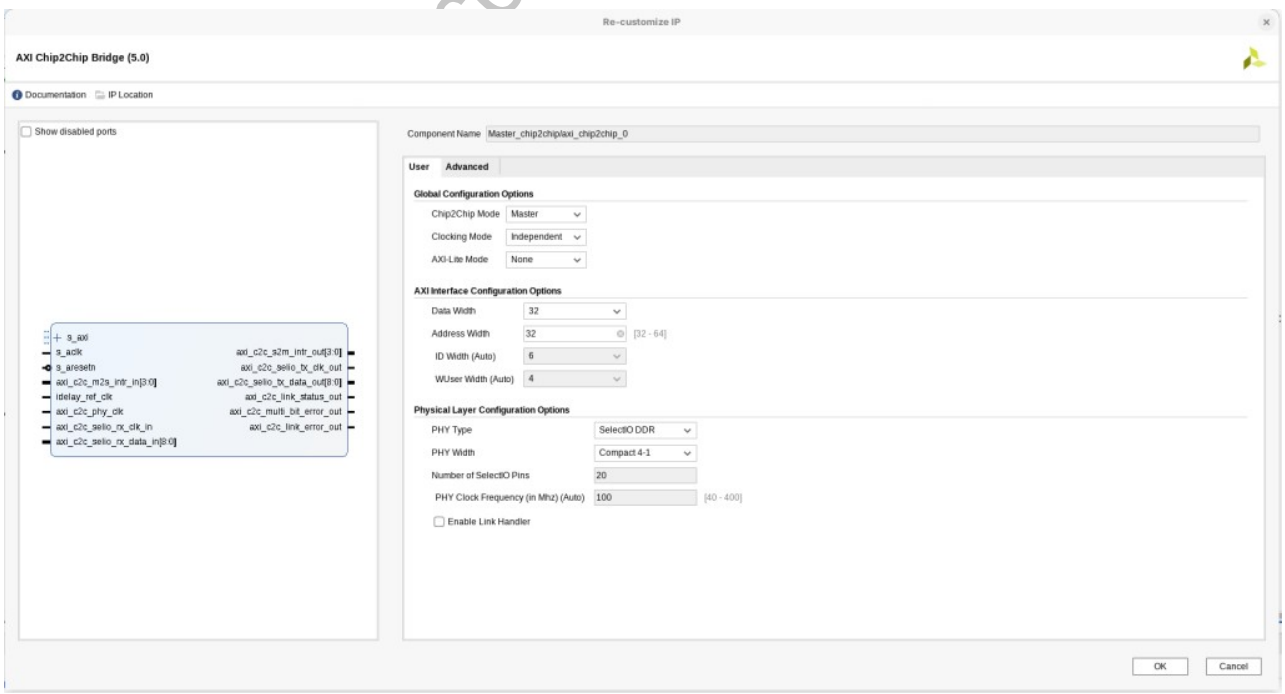
NOTA: (importante) los puertos *axi_c2c_m2s_intr_in* y el *axi_c2c_s2m_intr_out* son los puertos que comunican las interrupciones. El de entrada es para la expansora y el de salida para la Zynq, que lo llevará junto al resto de interrupciones de la Zynq. La interrupción tiene que estar lo suficiente activa para poder transmitirse.

NOTA2: las señales de interrupción anteriores es posible que estén mal creadas en Vivado, por lo que no es recomendable fiarse del número de bits que ponen (siempre ponen [3:0]) porque se puede meter un *concat* con más y con menos señales que 4.

Para configurar este bloque se hace de la siguiente forma:

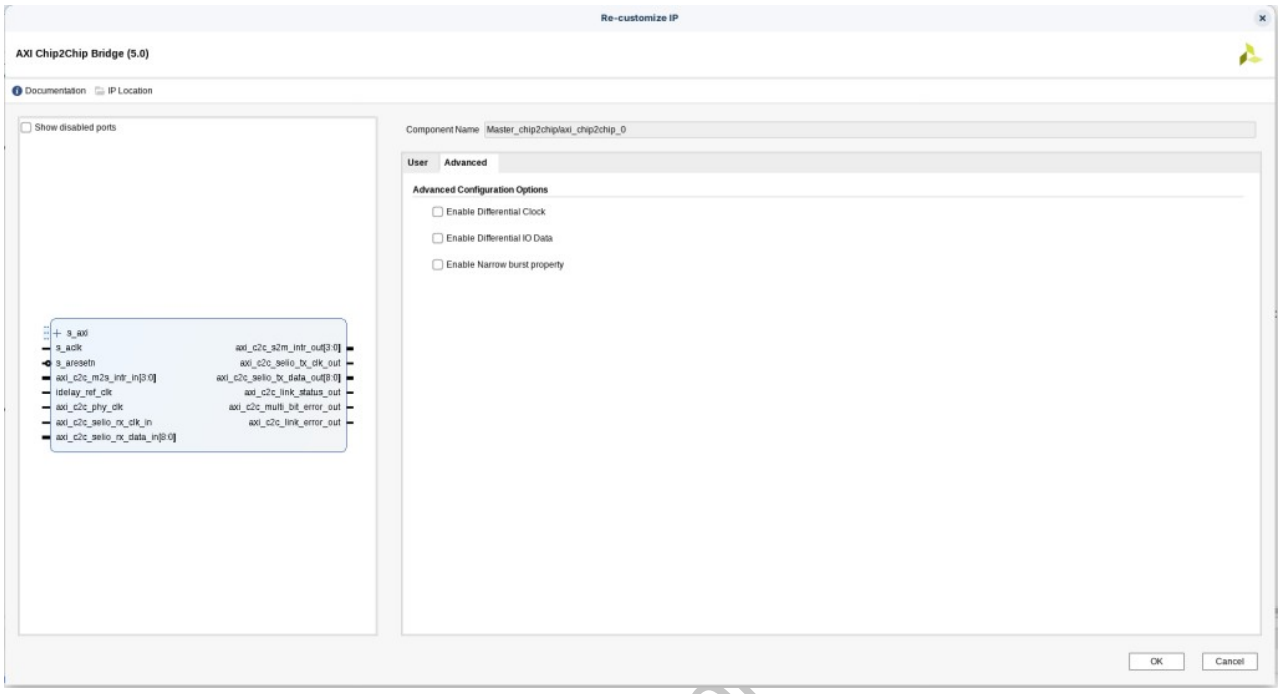
- **Pestaña User**

En esta pestaña se configura el bloque como Master, debido a que este Chip2Chip es el maestro de la comunicación. También se estable que el tamaño de los datos será de 32 bits y que la forma de comunicación es mediante **SelectIO DDR** con las líneas compactadas en 4:1 (es significa que el número de puertos necesarios para la comunicación se reducen)



- **Advanced**

La otra pestaña simplemente para configurar puertos diferenciales y de control.

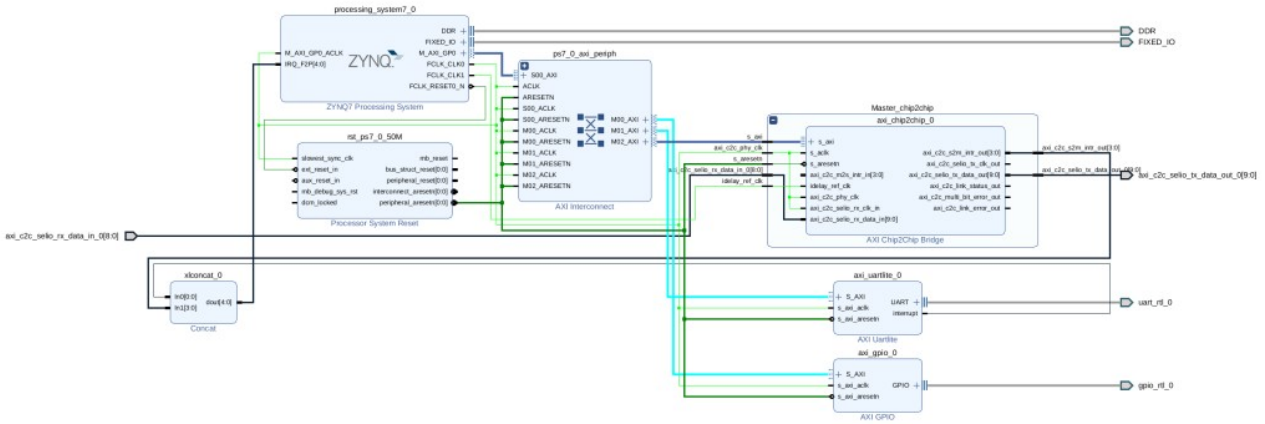


Es importante, a la hora de hacer el XDC, declarar los pines del *SelectIO* (cosa que con lo pines GT no se tiene que hacer).

NOTA: para definir los pines del *SelectIO*, estos tienen que estar en un banco alimentado a 2.5V, incluido el reloj.

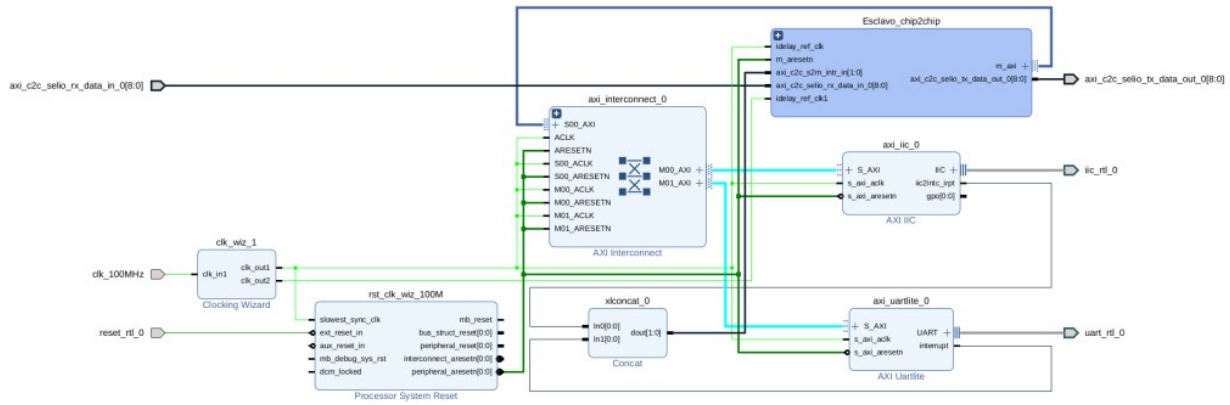
```
set_property PACKAGE_PIN G12 [get_ports reset]
set_property IOSTANDARD LVCMOS25 [get_ports reset]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_clk_out_slv]
set_property LOC F20 [ get_ports t_axi_sio_clk_out_slv]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_clk_in_slv]
set_property LOC D12 [ get_ports t_axi_sio_clk_in_slv]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_data_out_slv[*]]
set_property LOC H25 [ get_ports t_axi_sio_data_out_slv[0]]
set_property LOC H24 [ get_ports t_axi_sio_data_out_slv[1]]
set_property LOC H27 [ get_ports t_axi_sio_data_out_slv[2]]
set_property LOC H26 [ get_ports t_axi_sio_data_out_slv[3]]
set_property LOC F28 [ get_ports t_axi_sio_data_out_slv[4]]
set_property LOC G28 [ get_ports t_axi_sio_data_out_slv[5]]
set_property LOC F30 [ get_ports t_axi_sio_data_out_slv[6]]
set_property LOC G29 [ get_ports t_axi_sio_data_out_slv[7]]
set_property LOC G30 [ get_ports t_axi_sio_data_out_slv[8]]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_data_in_slv[*]]
set_property LOC C11 [ get_ports t_axi_sio_data_in_slv[0]]
set_property LOC D11 [ get_ports t_axi_sio_data_in_slv[1]]
set_property LOC B12 [ get_ports t_axi_sio_data_in_slv[2]]
set_property LOC C12 [ get_ports t_axi_sio_data_in_slv[3]]
set_property LOC E11 [ get_ports t_axi_sio_data_in_slv[4]]
set_property LOC F11 [ get_ports t_axi_sio_data_in_slv[5]]
set_property LOC E16 [ get_ports t_axi_sio_data_in_slv[6]]
set_property LOC F15 [ get_ports t_axi_sio_data_in_slv[7]]
set_property LOC C14 [ get_ports t_axi_sio_data_in_slv[8]]
```

La forma final del diagrama queda de la siguiente forma



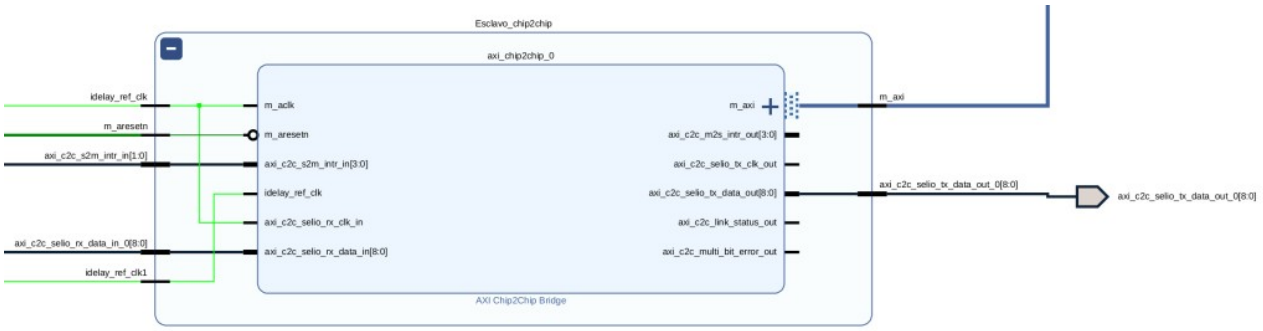
Configuración de la Expansora

Para realizar la configuración de la expansora tenemos que tener claro, que esta va a ser esclava del *AXI_Chip2chip*, pero que va a ser la maestra del *Axi_interconnect*.



Como queda reflejado en la imagen anterior, se puede ver que la forma de conectar el *AXI_chip2chip* es la misma que en la Zynq.

Si abrimos el bloque, se puede ver que ahora las interrupciones son de entrada por el *axi_c2c_s2m_intr_in*. El resto es igual que la Zynq.



https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/

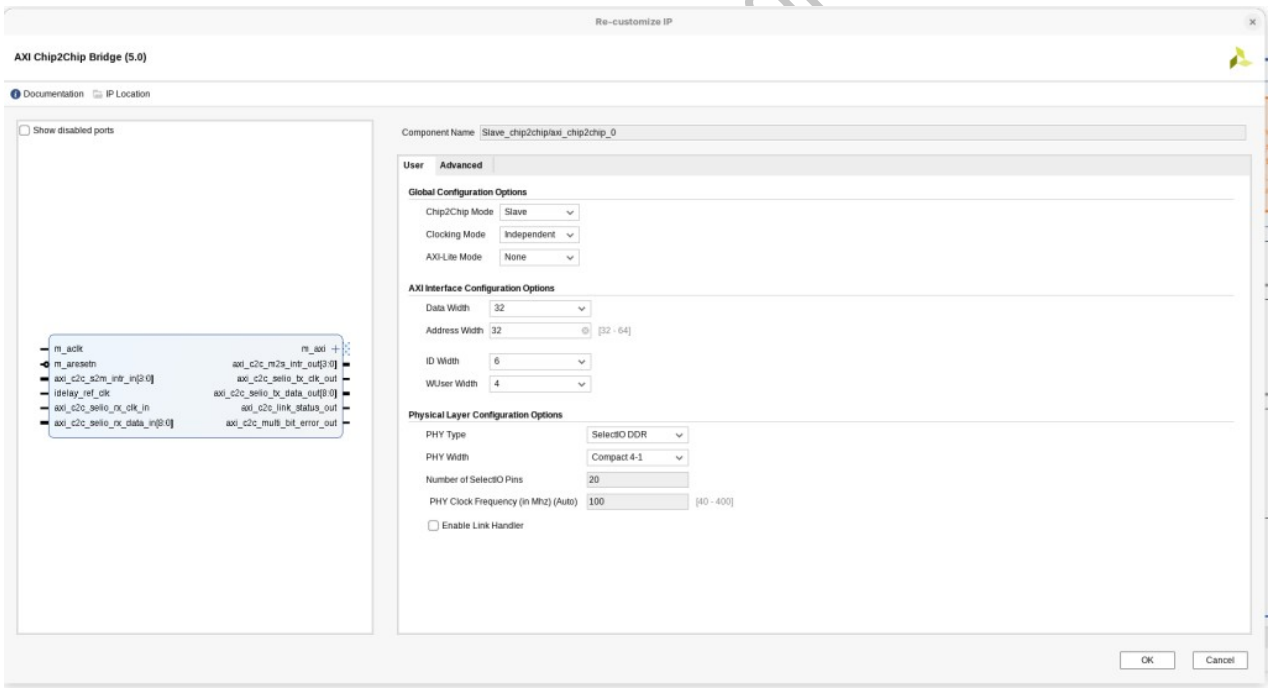
NOTA: (importante) los puertos *axi_c2c_m2s_intr_in* y el *axi_c2c_s2m_intr_out* son los puertos que comunican las interrupciones. El de entrada es para la expansora y el de salida para la Zynq, que lo llevará junto al resto de interrupciones de la Zynq. La interrupción tiene que estar lo suficiente activa para poder transmitirse.

NOTA2: las señales de interrupción anteriores es posible que estén mal creadas en Vivado, por lo que no es recomendable fiarse del número de bits que ponen (siempre ponen [3:0]) porque se puede meter un *concat* con más y con menos señales que 4.

NOTA 3: el puerto de entrada del *AXI_chip2chip*, **idelay_ref_clk**, tiene que estar a un reloj de 200MHz o de 300MHz, si no genera un error. Este reloj puede venir del exterior o de un PLL interno. *(en versiones antiguas de Vivado esta señal se puede desactivar utilizando la propiedad de Des-skew)*

idelay_ref_clk	Input	SelectIO™ Interface I/O Reference Clock. This signal is applicable only when the SelectIO™ interface is selected as the FPGA interfacing option. The applicable frequency for idelay_ref_clk is 200 MHz or 300 MHz (±10 MHz).

La configuración interna del bloque queda de la forma siguiente:

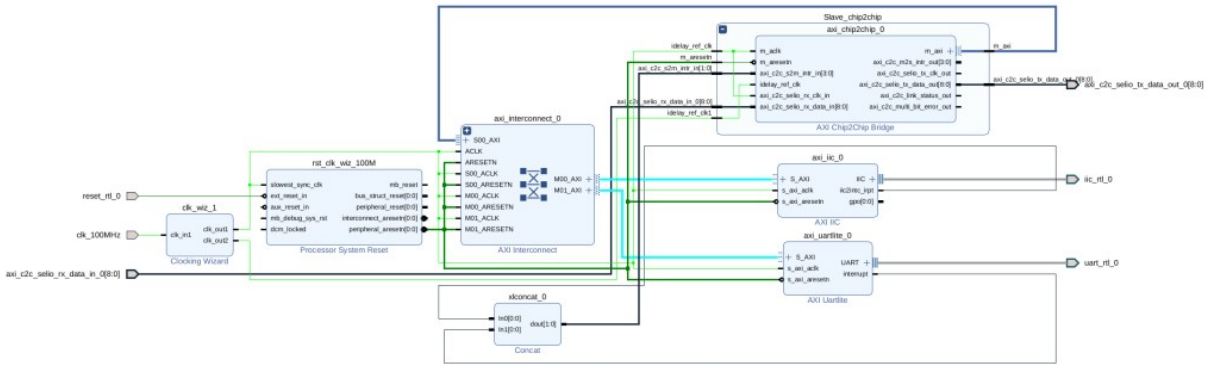


La configuración interna es igual que la de la Zynq, solo que esta es esclavo del *AXI_chip2chip*, lo que significa que es el master del AXI del *AXI_interconnect*.

NOTA: para definir los pines del *SelectIO*, estos tienen que estar en un banco alimentado a 2.5V, incluido el reloj.

```
set_property PACKAGE_PIN G12 [get_ports reset]
set_property IOSTANDARD LVCMOS25 [get_ports reset]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_clk_out_slv]
set_property LOC F20 [ get_ports t_axi_sio_clk_out_slv]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_clk_in_slv]
set_property LOC D12 [ get_ports t_axi_sio_clk_in_slv]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_data_out_slv[*]]
set_property LOC H25 [ get_ports t_axi_sio_data_out_slv[0]]
set_property LOC H24 [ get_ports t_axi_sio_data_out_slv[1]]
set_property LOC H27 [ get_ports t_axi_sio_data_out_slv[2]]
set_property LOC H26 [ get_ports t_axi_sio_data_out_slv[3]]
set_property LOC F28 [ get_ports t_axi_sio_data_out_slv[4]]
set_property LOC G28 [ get_ports t_axi_sio_data_out_slv[5]]
set_property LOC F30 [ get_ports t_axi_sio_data_out_slv[6]]
set_property LOC G29 [ get_ports t_axi_sio_data_out_slv[7]]
set_property LOC G30 [ get_ports t_axi_sio_data_out_slv[8]]
set_property IOSTANDARD LVCMOS25 [get_ports t_axi_sio_data_in_slv[*]]
set_property LOC C11 [ get_ports t_axi_sio_data_in_slv[0]]
set_property LOC D11 [ get_ports t_axi_sio_data_in_slv[1]]
set_property LOC B12 [ get_ports t_axi_sio_data_in_slv[2]]
set_property LOC C12 [ get_ports t_axi_sio_data_in_slv[3]]
set_property LOC E11 [ get_ports t_axi_sio_data_in_slv[4]]
set_property LOC F11 [ get_ports t_axi_sio_data_in_slv[5]]
set_property LOC E16 [ get_ports t_axi_sio_data_in_slv[6]]
set_property LOC F15 [ get_ports t_axi_sio_data_in_slv[7]]
set_property LOC C14 [ get_ports t_axi_sio_data_in_slv[8]]
```

La forma final del diagrama queda de la siguiente forma.



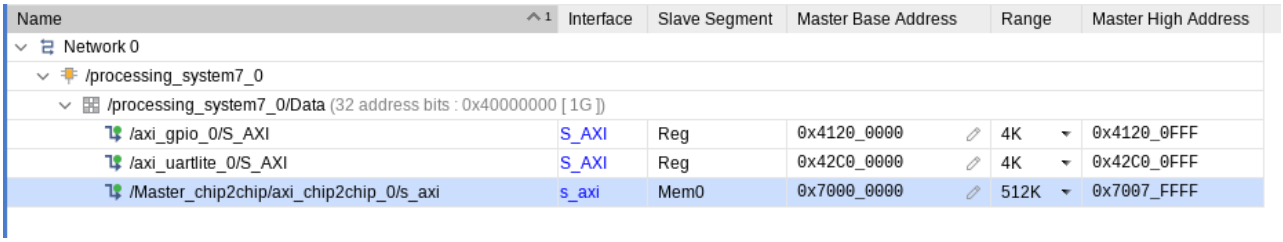
Direcciones de Memoria

Esta es una parte importante porque como he comentado anteriormente, la Zynq tiene que reservar un espacio de memoria que será de uso exclusivo por la expansora.

- **Direcciones de la Zynq**

En estas direcciones se ha decidido reservar un espacio de 512kB a partir de la dirección 0x7000_0000. Este es el espacio para la expansora.

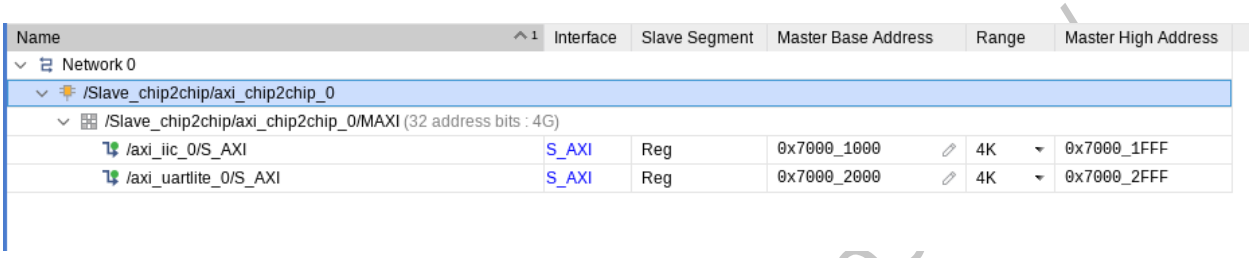
https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x4120_0000	4K	0x4120_0FFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x42C0_0000	4K	0x42C0_0FFF
/Master_chip2chip/axi_chip2chip_0/s_axi	s_axi	Mem0	0x7000_0000	512K	0x7007_FFFF

- **Direcciones de la expansora**

Las direcciones de memoria de la expansora comienzan desde la dirección 0x7000_0000 y no exceden los 512kB reservados.



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/Slave_chip2chip/axi_chip2chip_0					
/Slave_chip2chip/axi_chip2chip_0/MAXI (32 address bits : 4G)					
/axi_iic_0/S_AXI	S_AXI	Reg	0x7000_1000	4K	0x7000_1FFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x7000_2000	4K	0x7000_2FFF

NOTA: Es importante tener en cuenta en las direcciones de memoria a la hora de irse a Vitis *(antiguo Vivado SDK)* debido a que la Zynq sólo reconoce sus direcciones de memoria propias, es decir aquellas que están en el proyecto con la Zynq. Por lo que las direcciones de memoria de la expansora no van a estar disponibles en Vitis, porque nadie las va a generar en C. Entonces, tiene que ser el propio usuario quien incorpore las direcciones de memoria a mano en un fichero .h para tenerlas disponibles, y ser cuidadoso de que al añadir más bloques IP se generan nuevas direcciones de memoria.

Con todo esto ya quedarían conectadas la Zynq y la expansora a través del *SelectIO*.

Ejemplo 2

En este ejemplo se va a realizar lo mismo que en el anterior ejemplo, solo que ahora la Zynq y la expansora van a estar conectadas por pines GT. Para ello se hará uso del bloque IP Aurora 8b/10b.

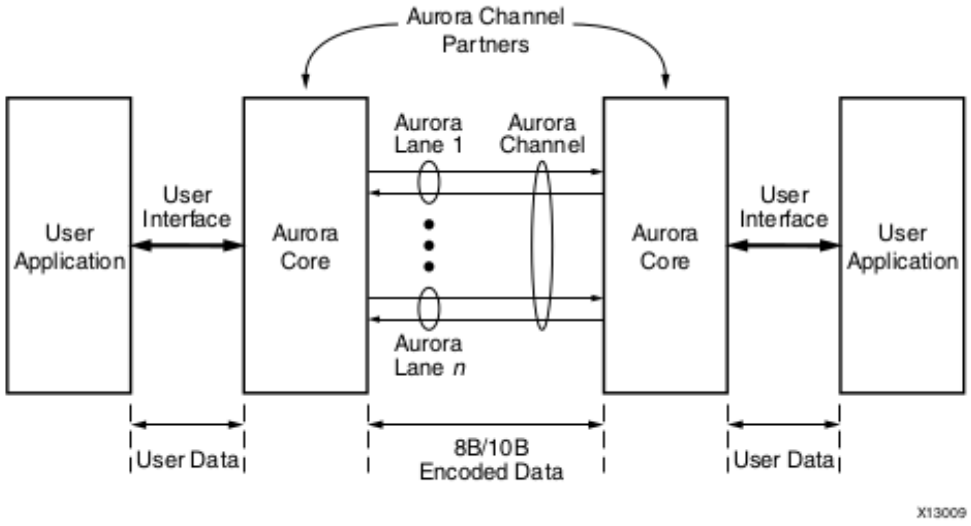


Figure 1-1: **Aurora 8B/10B Channel Overview**

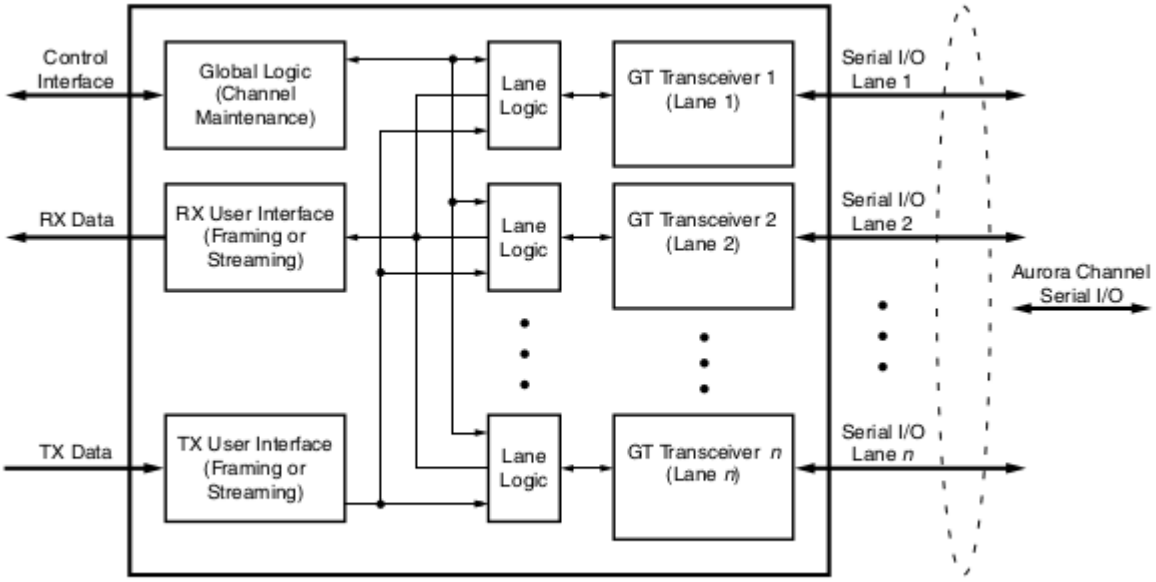
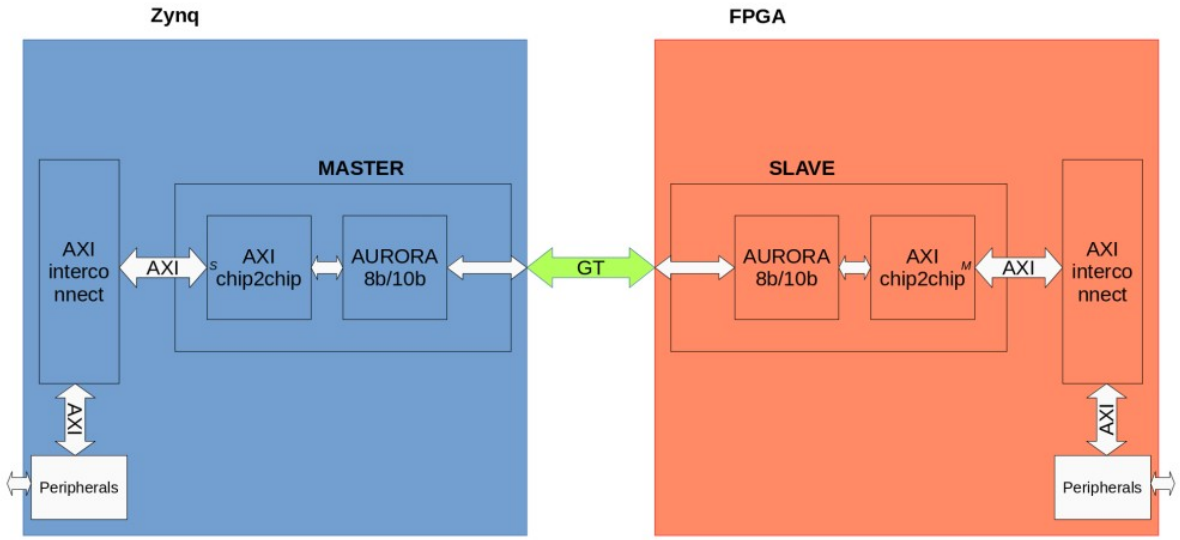


Figure 2-1: **Aurora 8B/10B Core Block Diagram**

El esquema de comunicación quedaría de la siguiente forma.

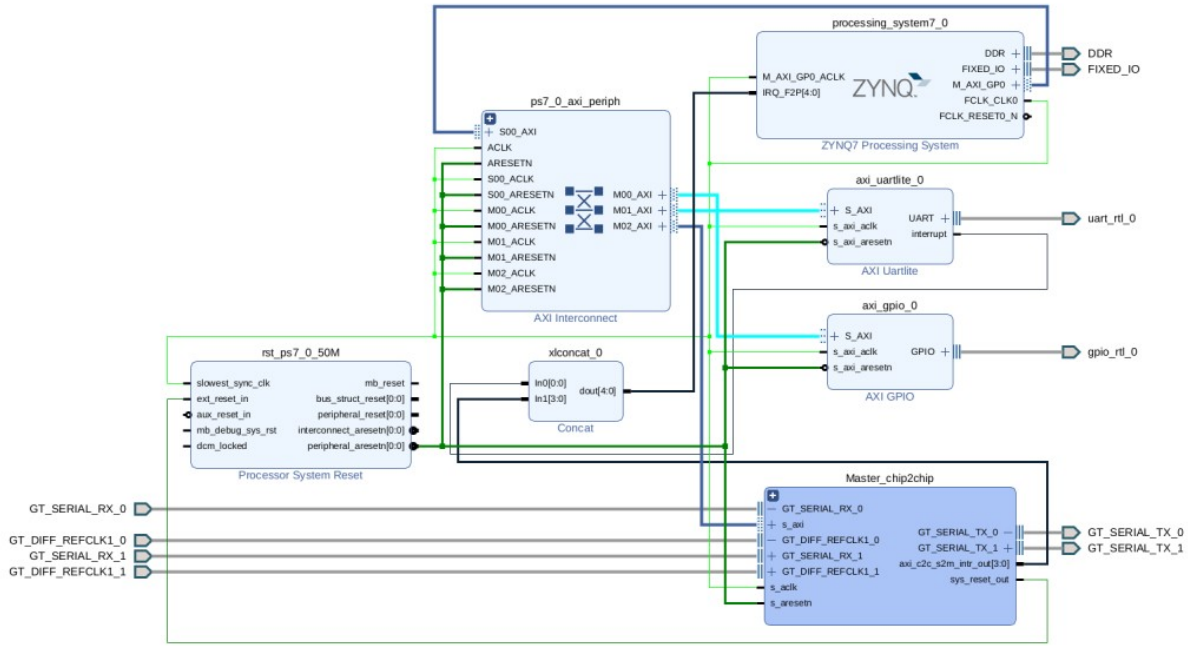


Para ello es importante conocer que la expansión con el AURORA significa que se van a utilizar líneas de tipo GT, cosa que no todas las Zynq's o las FPGAs de Xilinx poseen. Entonces, es importante saber elegir bien el dispositivo deseado.

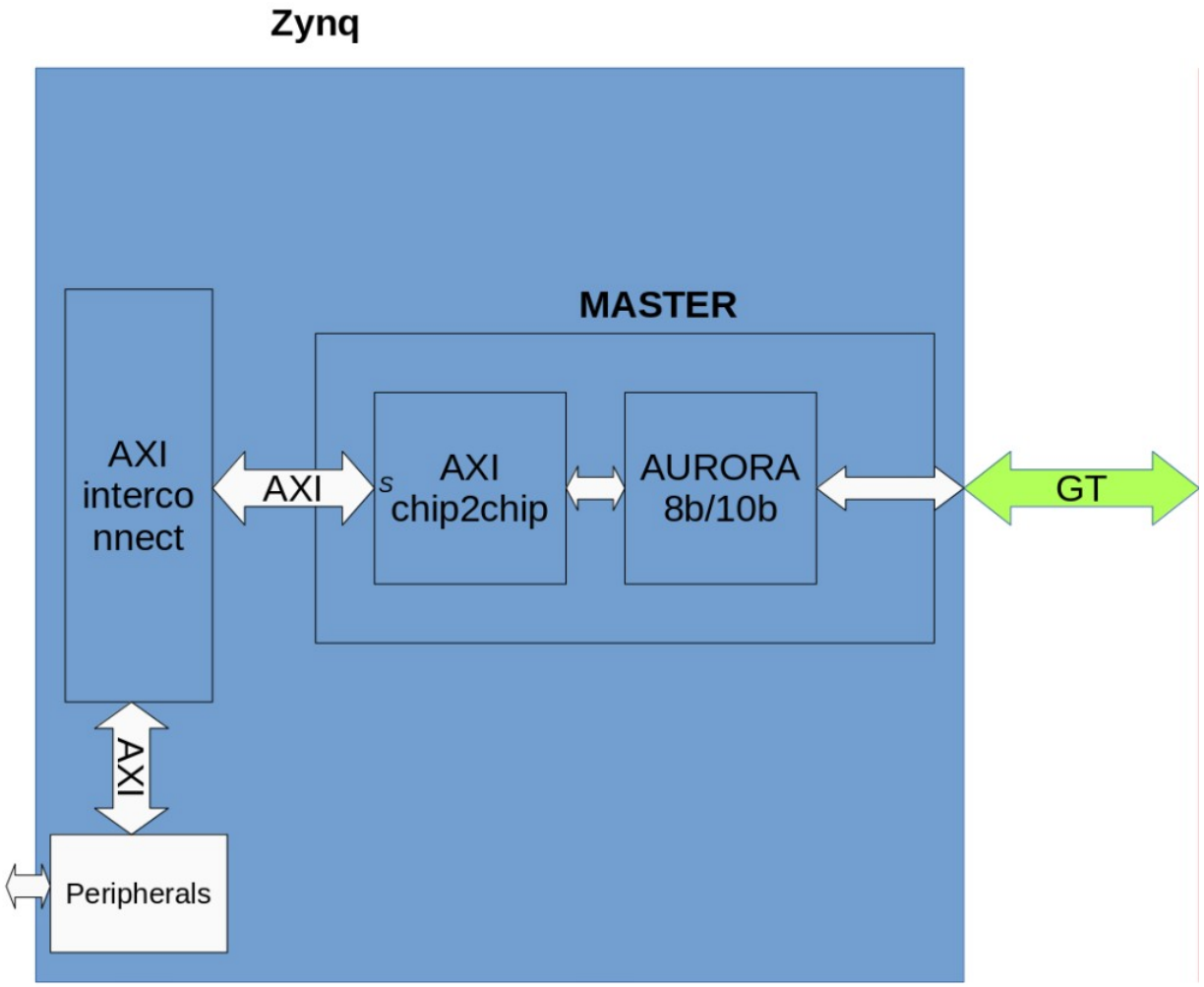
Este ejemplo es igual que el anterior. En este ejemplo tenemos una Zynq que se nos ha quedado corta de pines, y necesitamos todavía un bloque AXI_I2C y un AXI_UART.

Configuración de la Zynq

Para configurar la Zynq, primero configuramos una Zynq normal y corriente, y le añadimos un bloque que será el que se comunicará con la expansora.



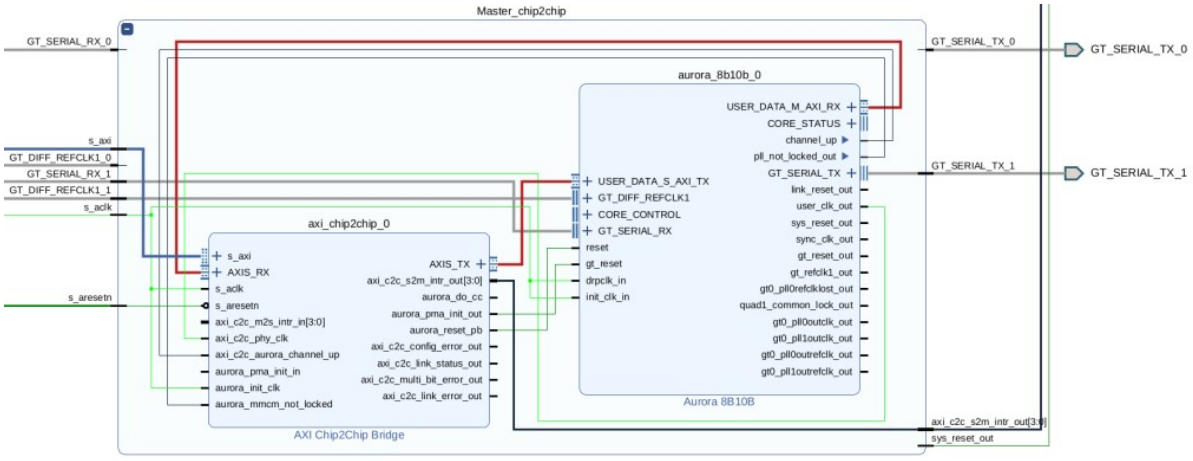
Este bloque AXI se configura utilizando dos bloques IP como si fuesen uno. Un *AXI_chip2chip*, que es el bloque IP que nos reservará las direcciones de memoria que se utilizarán en la FPGA, y el *Aurora8b/10b* que es el que hará la comunicación con la expansora a alta velocidad.



El *Axi_chip2chip* tiene que estar configurado como maestro de la comunicación, porque es el que transmite las líneas a la expansora, pero tiene que ser esclavo del *AXI interconnect*, porque los dispositivos de la expansora son esclavos de la Zynq.

Esto se consigue conectando los dos bloques IP de la siguiente forma, en la que la comunicación con la expansora se realiza por el bloque AURORA a través de las líneas **GT_SERIAL_TX_0** y **GT_SERIAL_RX_0**, que son líneas de tipo GT que se declaran internamente, o sea, que no aparecen estos puertos en el XDC. Y una línea de reloj diferencial, **GT_DIFF_REFCLK_0**, para el bloque Aurora (esta sí que aparece en el XDC.

https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/



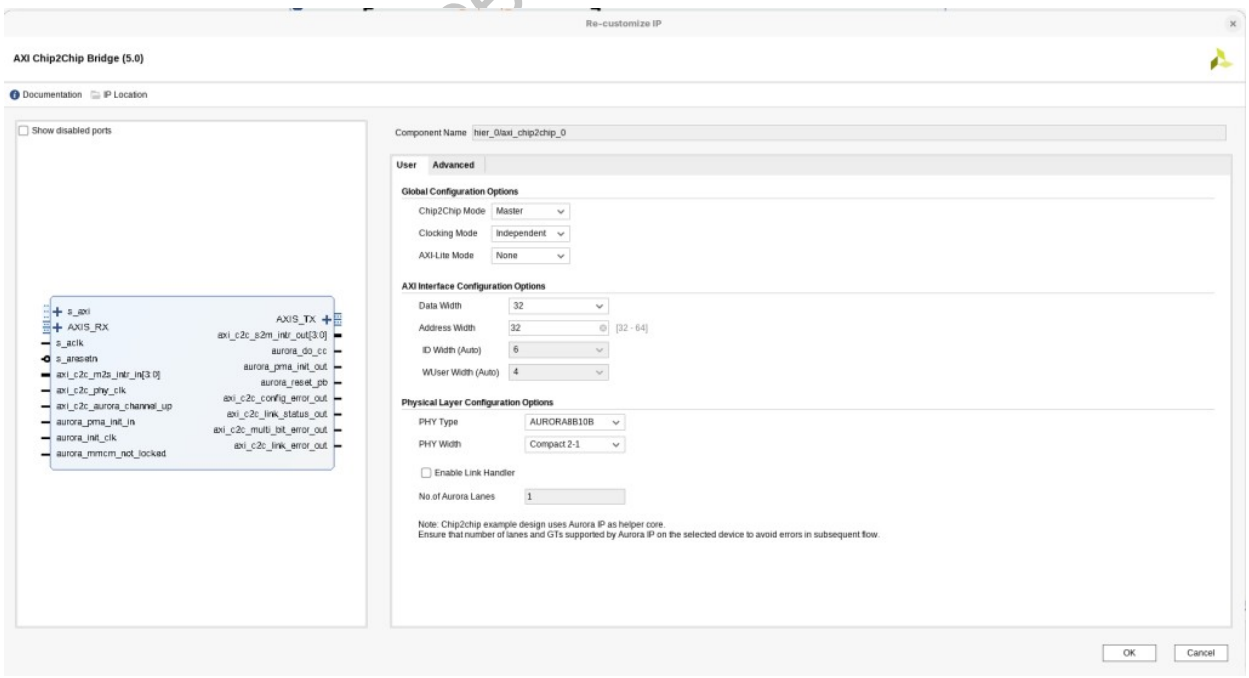
NOTA: (importante) los puertos *axi_c2c_m2s_intr_in* y el *axi_c2c_s2m_intr_out* son los puertos que comunican las interrupciones. El de entrada es para la expansora y el de salida para la Zynq, que lo llevará junto al resto de interrupciones de la Zynq. La interrupción tiene que estar lo suficiente activa para poder transmitirse.

NOTA2: las señales de interrupción anteriores es posible que estén mal creadas en Vivado, por lo que no es recomendable fiarse del número de bits que ponen (siempre ponen [3:0]) porque se puede meter un *concat* con más y con menos señales que 4.

Cada bloque se configura internamente de la siguiente manera.

- **AXI_chip2chip**

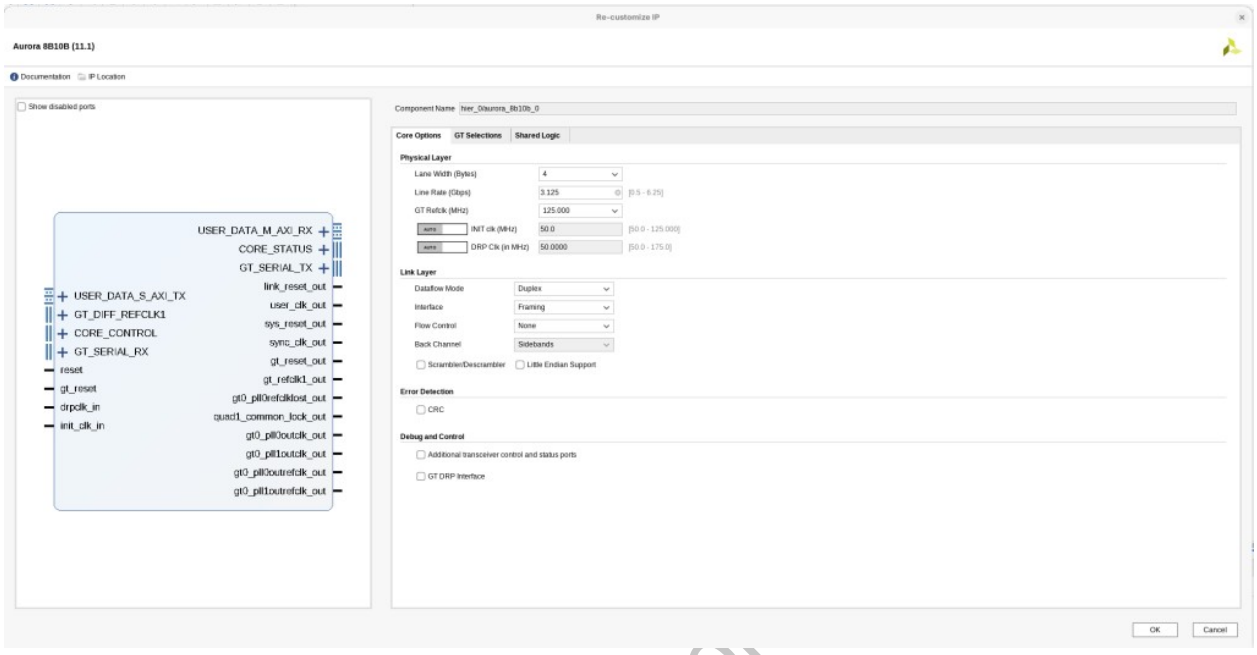
El Axi_chip2chip se configura como **master**, con un tamaño de datos de 32bits y un sistema de comunicación por **AURORA8B10B**. De ahí que se añade el bloque **AURORA_8B10B**.



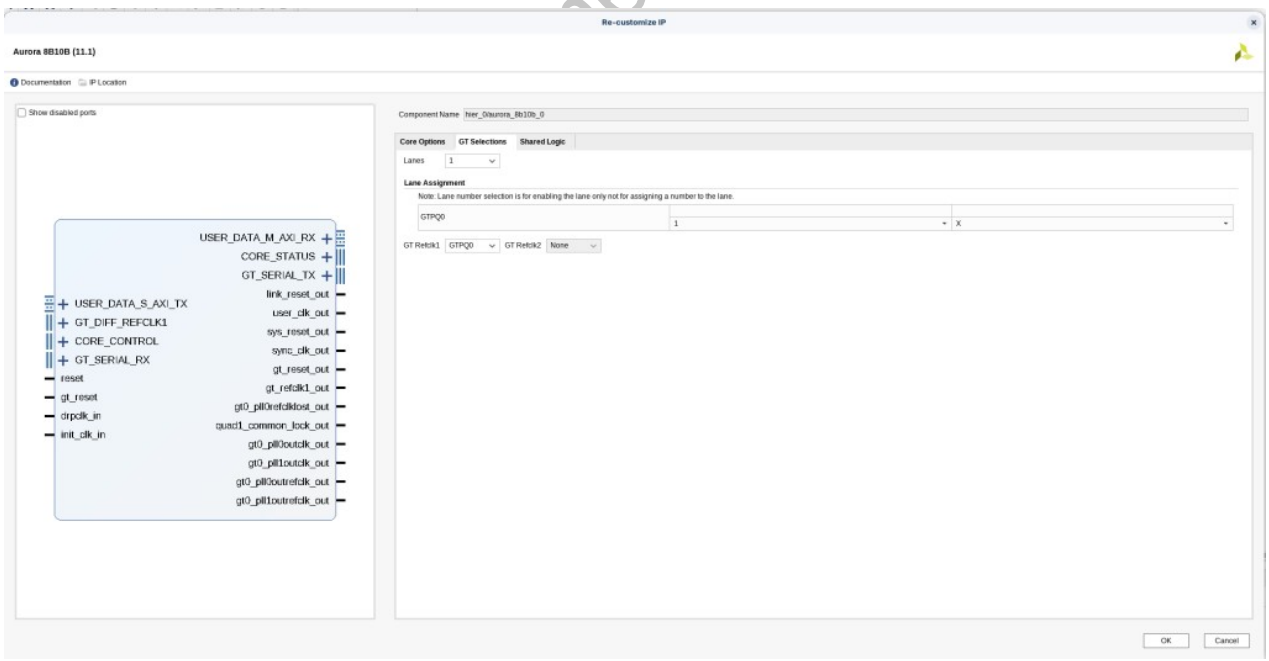
https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/

- **Aurora_8b10b**

El bloque Aurora se configura con 4 líneas de comunicación a 3.125Gbps, con un reloj de referencia de 125MHz. Este reloj de referencia tiene que venir del exterior y se declara en el XDC.



En la siguiente pestaña se establece cuáles de los transceivers se van a utilizar. En el caso de la Zynq que he elegido solo tiene un transceiver.

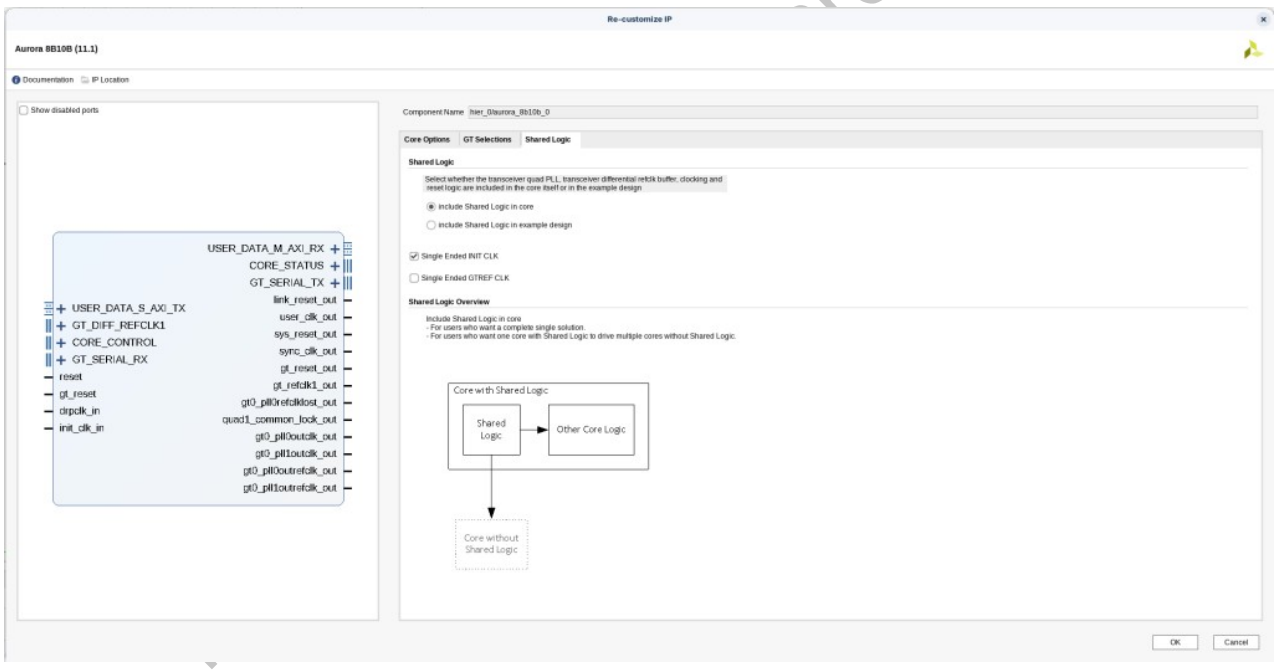


(en la siguiente imagen podemos ver que el xc7z015clg485 solo tiene un banco de transceivers, el 112, con 4 canales, que son los que utiliza el Aurora para comunicarse)

https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/

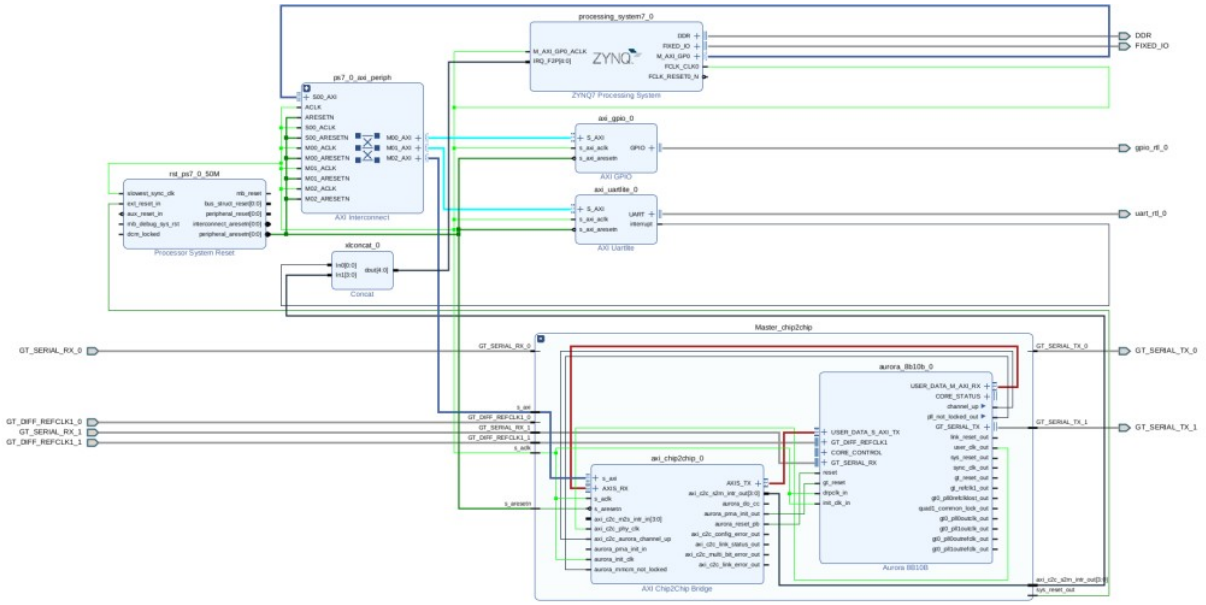
G1	IO_L24N_T3_AD15N_35	3	35	NA	NA	HR	NA
H5	IO_25_35	NA	35	NA	NA	HR	NA
W2	MGTPTXP3_112	NA	112	NA	NA	GTP	NA
Y2	MGTPTXN3_112	NA	112	NA	NA	GTP	NA
AA5	MGTPTXP2_112	NA	112	NA	NA	GTP	NA
AB5	MGTPTXN2_112	NA	112	NA	NA	GTP	NA
U9	MGTREFCLK0P_112	NA	112	NA	NA	GTP	NA
V9	MGTREFCLK0N_112	NA	112	NA	NA	GTP	NA
AA9	MGTPRXP2_112	NA	112	NA	NA	GTP	NA
W8	MGTPRXP1_112	NA	112	NA	NA	GTP	NA
AA7	MGTPRXP0_112	NA	112	NA	NA	GTP	NA
W6	MGTPRXP3_112	NA	112	NA	NA	GTP	NA
AA3	MGTPTXP0_112	NA	112	NA	NA	GTP	NA
W4	MGTPTXP1_112	NA	112	NA	NA	GTP	NA
U7	MGTRREF_112	NA	112	NA	NA	GTP	NA
V5	MGTREFCLK1N_112	NA	112	NA	NA	GTP	NA
U5	MGTREFCLK1P_112	NA	112	NA	NA	GTP	NA
AB9	MGTPRXN2_112	NA	112	NA	NA	GTP	NA
Y8	MGTPRXN1_112	NA	112	NA	NA	GTP	NA
AB7	MGTPRXN0_112	NA	112	NA	NA	GTP	NA
Y6	MGTPRXN3_112	NA	112	NA	NA	GTP	NA
AB3	MGTPTXN0_112	NA	112	NA	NA	GTP	NA
Y4	MGTPTXN1_112	NA	112	NA	NA	GTP	NA
F16	PS_CLK_500	NA	500	NA	NA	MIO	NA
F15	PS_MIO_VREF_501	NA	501	NA	NA	MIO	NA
B18	PS_POR_B_500	NA	500	NA	NA	MIO	NA

Y en la última pestaña, le decimos que utilice lógica compartida en el núcleo.



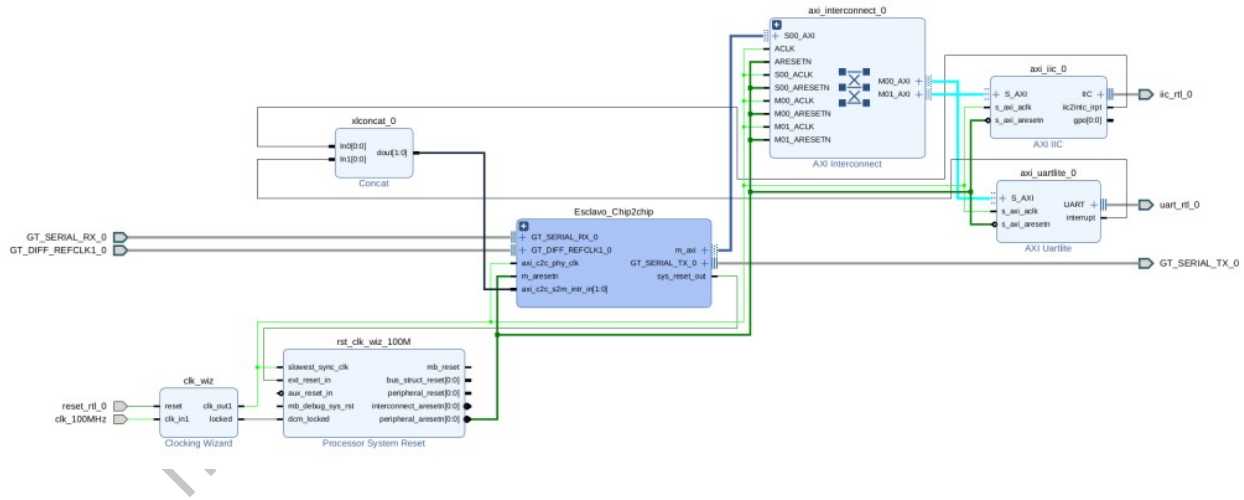
El modelo final de la Zynq queda de la siguiente forma.

https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/



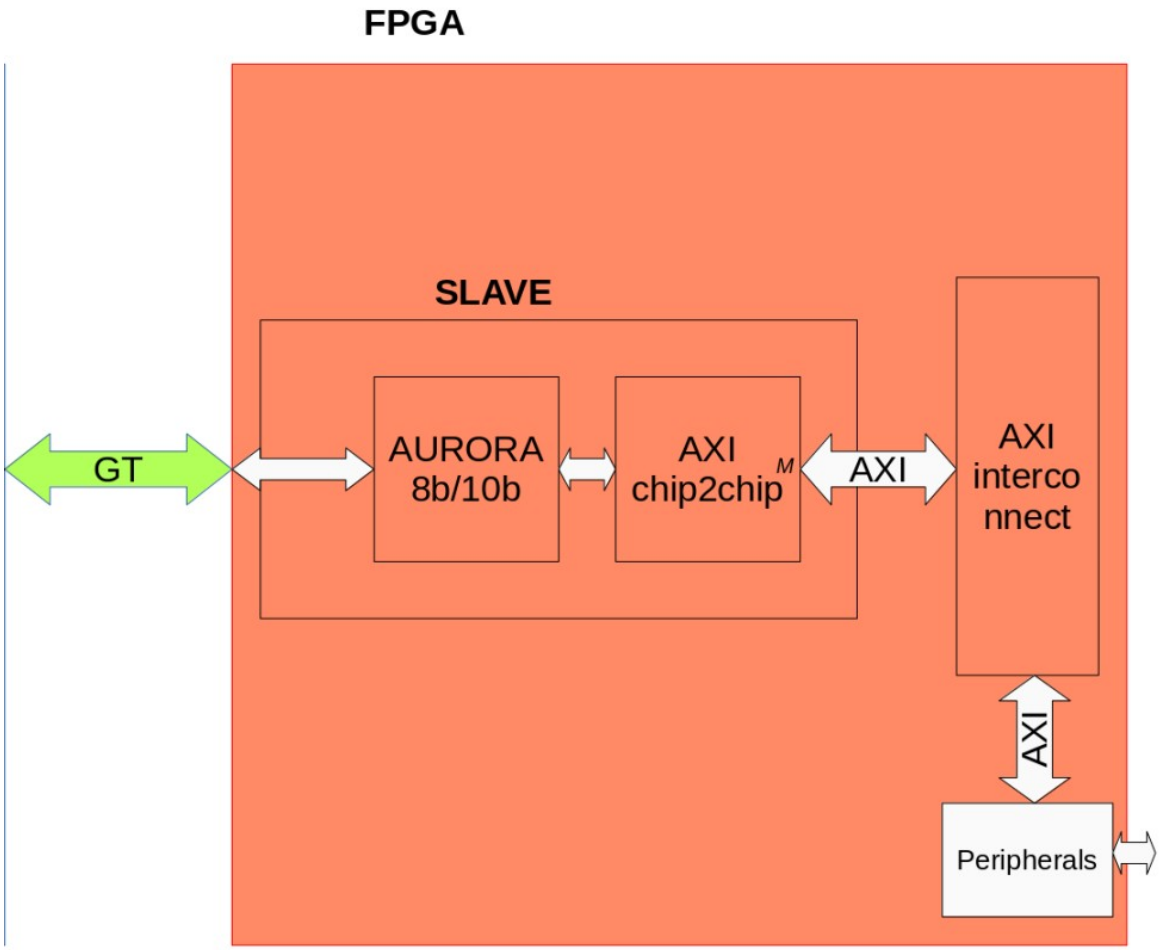
Configuración de la expansora

En el siguiente diagrama se puede ver un bloque IP que es el maestro del bloque *AXI_interconnect*, junto con los bloques IP deseados siendo esclavos del *AXI_interconnect*.

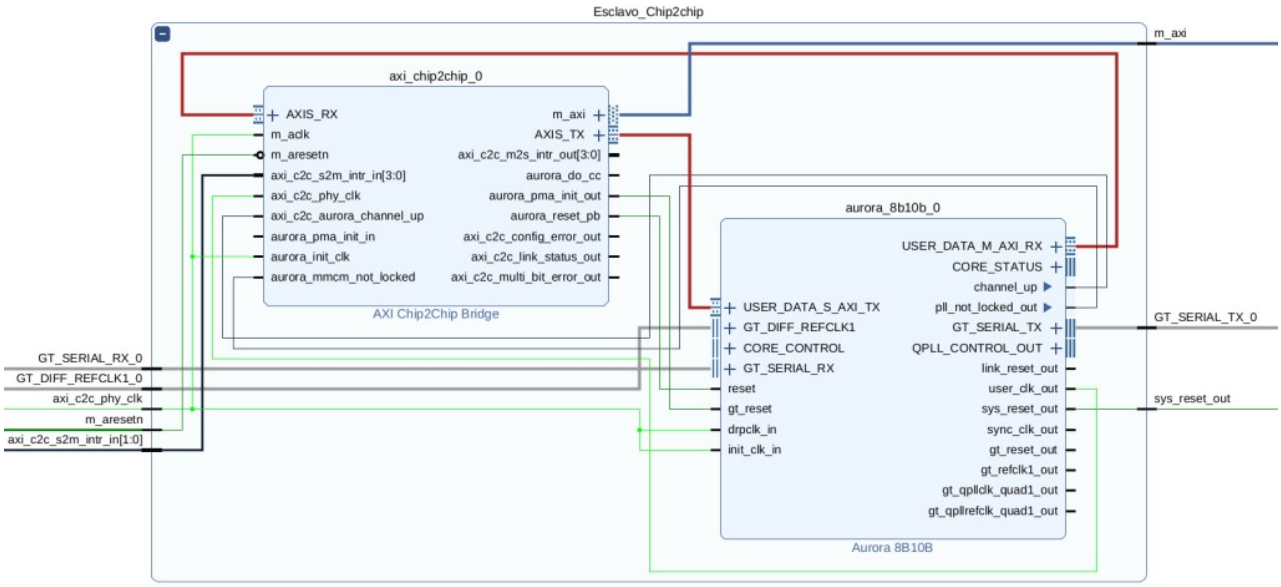


La comunicación con el AXI la realiza el *AXI_chip2chip* siendo maestro del bloque AXI interconnect. Pero, también el *AXI_chip2chip* es esclavo de la comunicación con la Zynq.

Para la configuración de la comunicación con el *AXI_chip2chip* se utiliza un esquema como este.



La conexión interna de los bloques queda de la siguiente forma. Siendo las líneas *GT_SERIAL_TX_0* y *GT_SERIAL_RX_0* las líneas de comunicación con la Zynq.



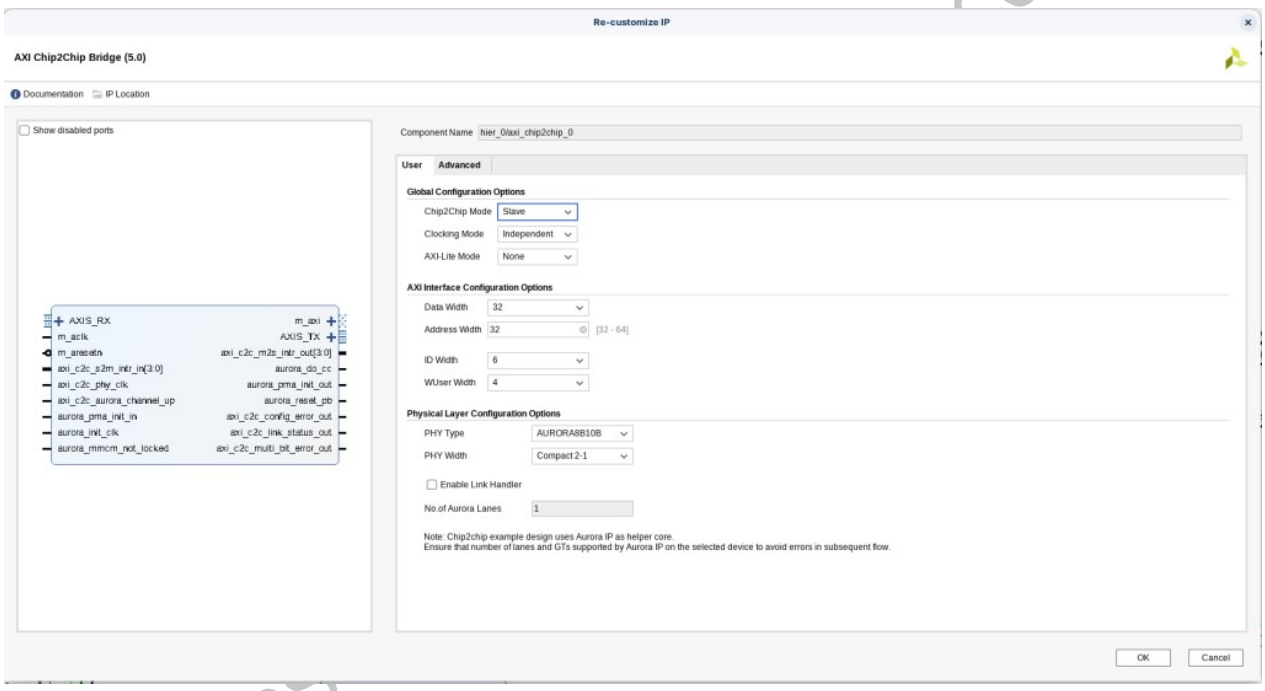
NOTA: (importante) los puertos *axi_c2c_m2s_intr_in* y el *axi_c2c_s2m_intr_out* son los puertos que comunican las interrupciones. El de entrada es para la expansora y el de salida para la Zynq, que lo llevará junto al resto de interrupciones de la Zynq. La interrupción tiene que estar lo suficiente activa para poder transmitirse.

NOTA2: las señales de interrupción anteriores es posible que estén mal creadas en Vivado, por lo que no es recomendable fiarse del número de bits que ponen (siempre ponen [3:0]) porque se puede meter un *concat* con más y con menos señales que 4.

La configuración interna de los bloques se realiza de la siguiente manera.

- **AXI_chip2chip**

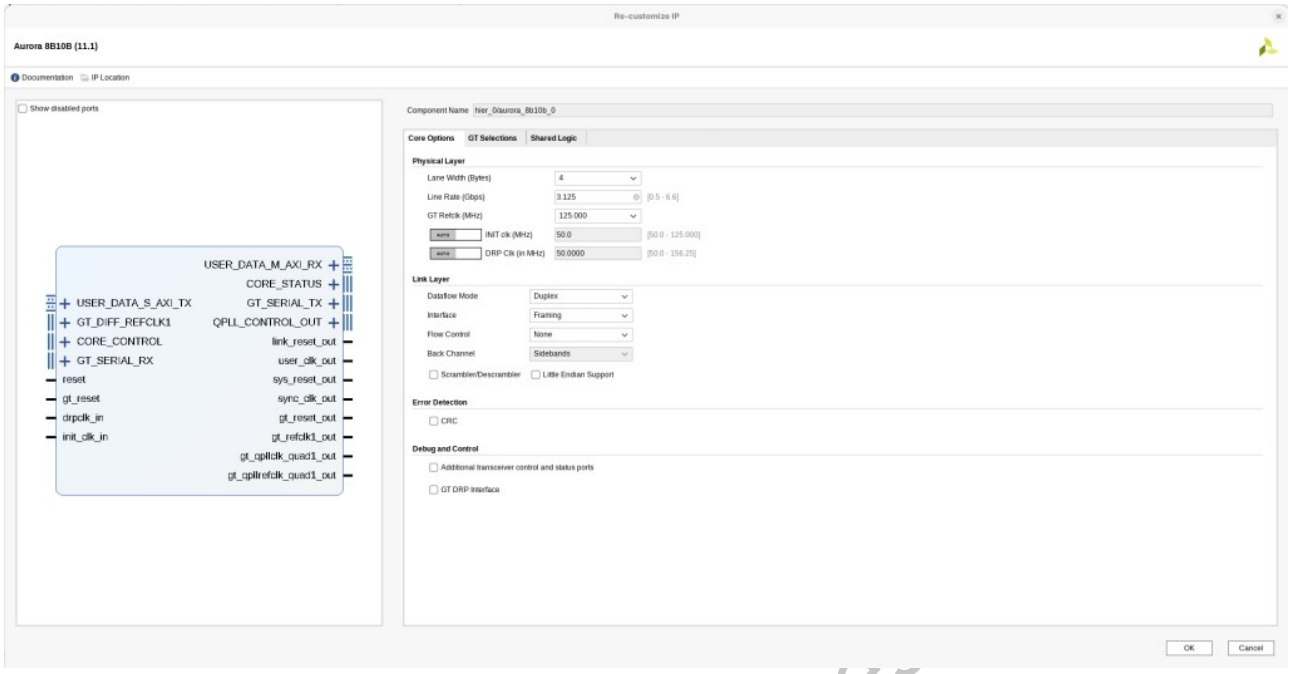
El *Axi_chip2chip* se tiene que configurar como esclavo de la comunicación, eso hace que sea maestro del bus AXI. También, se configura con 32 bits de datos y utilizando el *AURORA8B10B*.



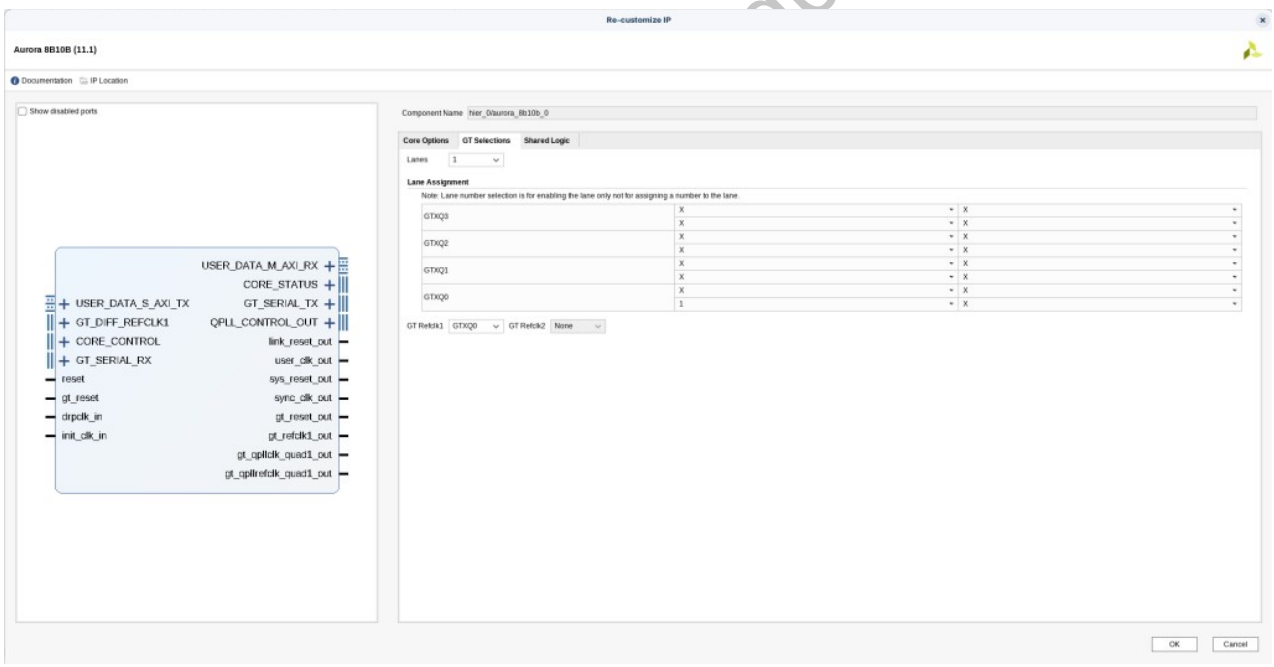
- **Aurora_8b10b**

Para la comunicación con la Zynq se van a utilizar 3.125Gbps en 4 líneas. Para conseguirlo se utiliza un reloj de 125MHz.

https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/



Para la comunicación se utiliza el banco 3 de los transceivers.



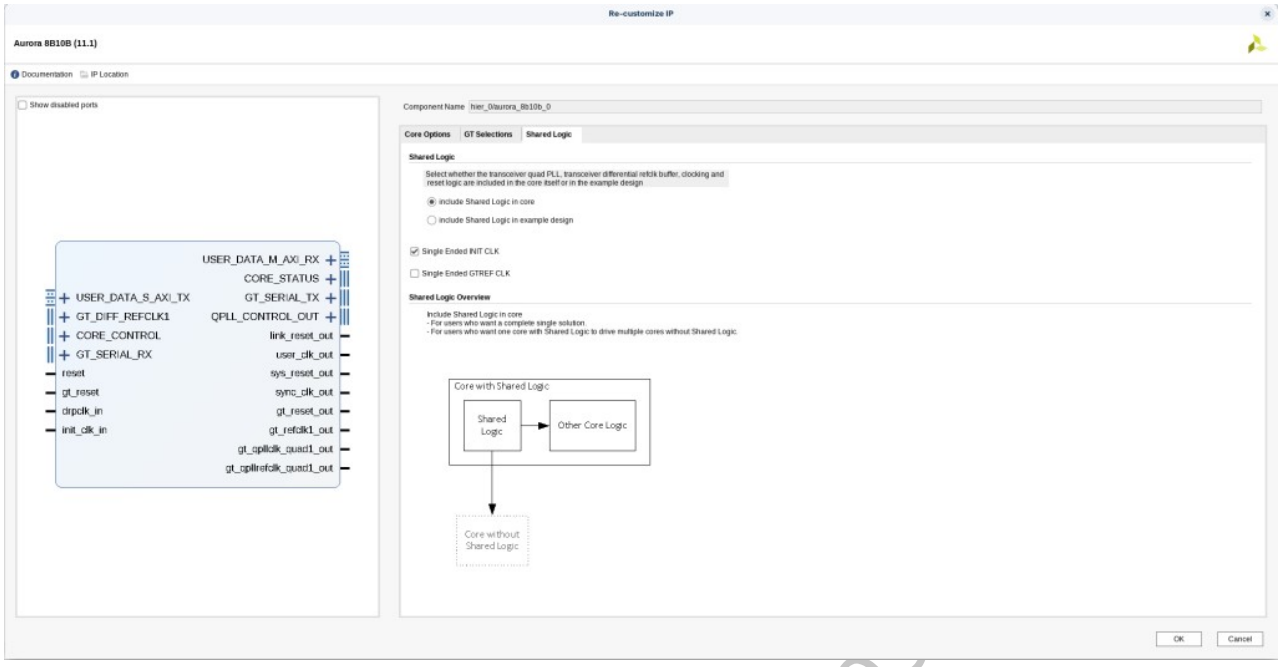
(en la siguiente imagen podemos ver que el xc7k325tffv900 tiene 4 bancos de transceivers, el 115, el 116, el 117 y el 118, en nuestro caso utilizamos el 118 que es el banco 3)

https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/

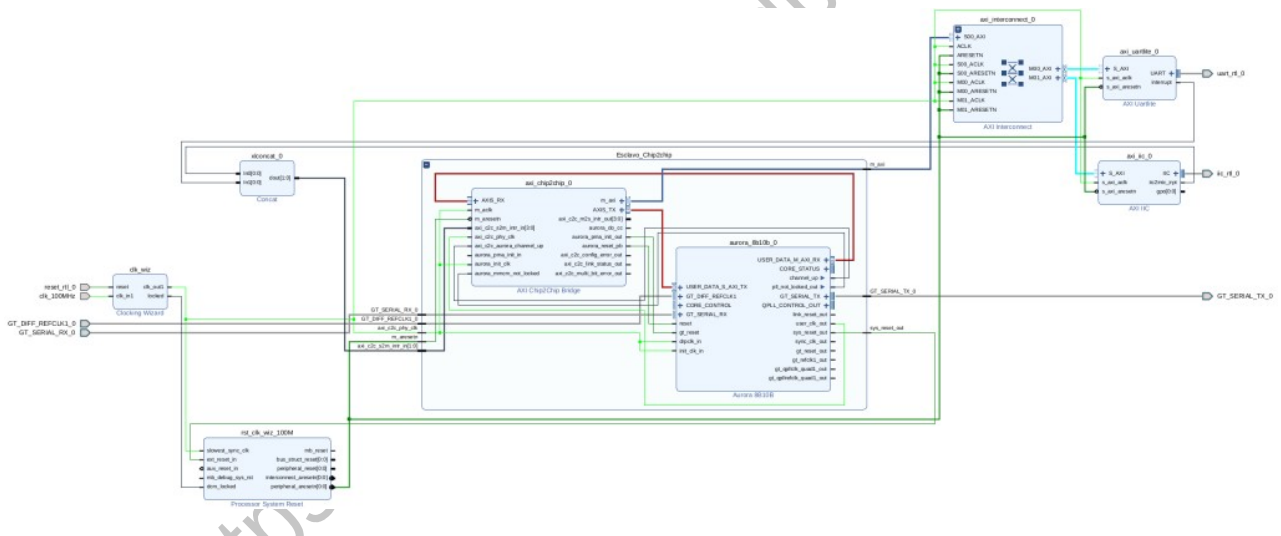
AK4	IO_L24N_T3_34	3	34	VCCAUX	NA	HP	NA
AB7	IO_25_VRP_34	NA	34	VCCAUX	NA	HP	NA
T2	MGTXTXP3_115	NA	115	NA	NA	GTX	NA
V6	MGTXRXP3_115	NA	115	NA	NA	GTX	NA
T1	MGTXTXN3_115	NA	115	NA	NA	GTX	NA
V5	MGTXRXN3_115	NA	115	NA	NA	GTX	NA
U4	MGTXTXP2_115	NA	115	NA	NA	GTX	NA
W4	MGTXRXP2_115	NA	115	NA	NA	GTX	NA
U3	MGTXTXN2_115	NA	115	NA	NA	GTX	NA
R8	MGTREFCLK0P_115	NA	115	NA	NA	GTX	NA
W3	MGTXRXN2_115	NA	115	NA	NA	GTX	NA
W7	MGTAVTTRCAL_115	NA	115	NA	NA	GTX	NA
R7	MGTREFCLK0N_115	NA	115	NA	NA	GTX	NA
W8	MGTRREF_115	NA	115	NA	NA	GTX	NA
U7	MGTREFCLK1N_115	NA	115	NA	NA	GTX	NA
U8	MGTREFCLK1P_115	NA	115	NA	NA	GTX	NA
V2	MGTXTXP1_115	NA	115	NA	NA	GTX	NA
Y6	MGTXRXP1_115	NA	115	NA	NA	GTX	NA
V1	MGTXTXN1_115	NA	115	NA	NA	GTX	NA
Y5	MGTXRXN1_115	NA	115	NA	NA	GTX	NA
Y2	MGTXTXP0_115	NA	115	NA	NA	GTX	NA
AA4	MGTXRXP0_115	NA	115	NA	NA	GTX	NA
Y1	MGTXTXN0_115	NA	115	NA	NA	GTX	NA
AA3	MGTXRXN0_115	NA	115	NA	NA	GTX	NA
L4	MGTXTXP3_116	NA	116	NA	NA	GTX	NA
M6	MGTXRXP3_116	NA	116	NA	NA	GTX	NA
L3	MGTXTXN3_116	NA	116	NA	NA	GTX	NA
M5	MGTXRXN3_116	NA	116	NA	NA	GTX	NA
M2	MGTXTXP2_116	NA	116	NA	NA	GTX	NA
P6	MGTXRXP2_116	NA	116	NA	NA	GTX	NA
M1	MGTXTXN2_116	NA	116	NA	NA	GTX	NA
L8	MGTREFCLK0P_116	NA	116	NA	NA	GTX	NA
P5	MGTXRXN2_116	NA	116	NA	NA	GTX	NA
L7	MGTREFCLK0N_116	NA	116	NA	NA	GTX	NA
N7	MGTREFCLK1N_116	NA	116	NA	NA	GTX	NA
N8	MGTREFCLK1P_116	NA	116	NA	NA	GTX	NA
N4	MGTXTXP1_116	NA	116	NA	NA	GTX	NA
R4	MGTXRXP1_116	NA	116	NA	NA	GTX	NA
N3	MGTXTXN1_116	NA	116	NA	NA	GTX	NA
R3	MGTXRXN1_116	NA	116	NA	NA	GTX	NA
P2	MGTXTXP0_116	NA	116	NA	NA	GTX	NA
T6	MGTXRXP0_116	NA	116	NA	NA	GTX	NA
P1	MGTXTXN0_116	NA	116	NA	NA	GTX	NA
T5	MGTXRXN0_116	NA	116	NA	NA	GTX	NA
F2	MGTXTXP3_117	NA	117	NA	NA	GTX	NA
F6	MGTXRXP3_117	NA	117	NA	NA	GTX	NA
F1	MGTXTXN3_117	NA	117	NA	NA	GTX	NA
F5	MGTXRXN3_117	NA	117	NA	NA	GTX	NA
H2	MGTXTXP2_117	NA	117	NA	NA	GTX	NA
G4	MGTXRXP2_117	NA	117	NA	NA	GTX	NA
H1	MGTXTXN2_117	NA	117	NA	NA	GTX	NA
G8	MGTREFCLK0P_117	NA	117	NA	NA	GTX	NA
G3	MGTXRXN2_117	NA	117	NA	NA	GTX	NA
G7	MGTREFCLK0N_117	NA	117	NA	NA	GTX	NA
J7	MGTREFCLK1N_117	NA	117	NA	NA	GTX	NA
J8	MGTREFCLK1P_117	NA	117	NA	NA	GTX	NA
J4	MGTXTXP1_117	NA	117	NA	NA	GTX	NA
H6	MGTXRXP1_117	NA	117	NA	NA	GTX	NA
J3	MGTXTXN1_117	NA	117	NA	NA	GTX	NA
H5	MGTXRXN1_117	NA	117	NA	NA	GTX	NA
K2	MGTXTXP0_117	NA	117	NA	NA	GTX	NA
K6	MGTXRXP0_117	NA	117	NA	NA	GTX	NA
K1	MGTXTXN0_117	NA	117	NA	NA	GTX	NA
K5	MGTXRXN0_117	NA	117	NA	NA	GTX	NA
A4	MGTXTXP3_118	NA	118	NA	NA	GTX	NA
A8	MGTXRXP3_118	NA	118	NA	NA	GTX	NA
A3	MGTXTXN3_118	NA	118	NA	NA	GTX	NA
A7	MGTXRXN3_118	NA	118	NA	NA	GTX	NA
B2	MGTXTXP2_118	NA	118	NA	NA	GTX	NA
B6	MGTXRXP2_118	NA	118	NA	NA	GTX	NA
B1	MGTXTXN2_118	NA	118	NA	NA	GTX	NA
C8	MGTREFCLK0P_118	NA	118	NA	NA	GTX	NA
B5	MGTXRXN2_118	NA	118	NA	NA	GTX	NA
C7	MGTREFCLK0N_118	NA	118	NA	NA	GTX	NA
E7	MGTREFCLK1N_118	NA	118	NA	NA	GTX	NA
E8	MGTREFCLK1P_118	NA	118	NA	NA	GTX	NA
C4	MGTXTXP1_118	NA	118	NA	NA	GTX	NA
D6	MGTXRXP1_118	NA	118	NA	NA	GTX	NA
C3	MGTXTXN1_118	NA	118	NA	NA	GTX	NA
D5	MGTXRXN1_118	NA	118	NA	NA	GTX	NA
D2	MGTXTXP0_118	NA	118	NA	NA	GTX	NA
E4	MGTXRXP0_118	NA	118	NA	NA	GTX	NA
D1	MGTXTXN0_118	NA	118	NA	NA	GTX	NA
E3	MGTXRXN0_118	NA	118	NA	NA	GTX	NA
B7	MGTAVCC	NA	NA	NA	NA	NA	NA
D7	MGTAVCC	NA	NA	NA	NA	NA	NA

Y en la última pestaña le decimos que utilice lógica compartida.

https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/



El diseño final de la expansora queda de la siguiente forma.



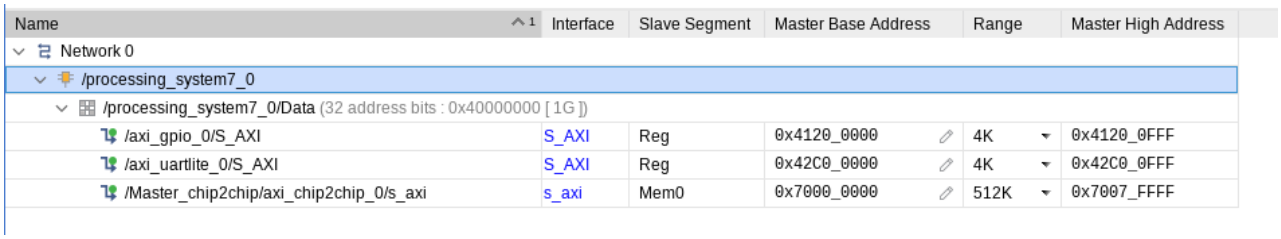
Direcciones de Memoria

Las direcciones de memoria de la Zynq y de la expansora quedan de la siguiente manera.

- **Direcciones de la Zynq**

Para la expansora se reservan 512kB para la expansora a partir de la dirección 0x7000_0000.

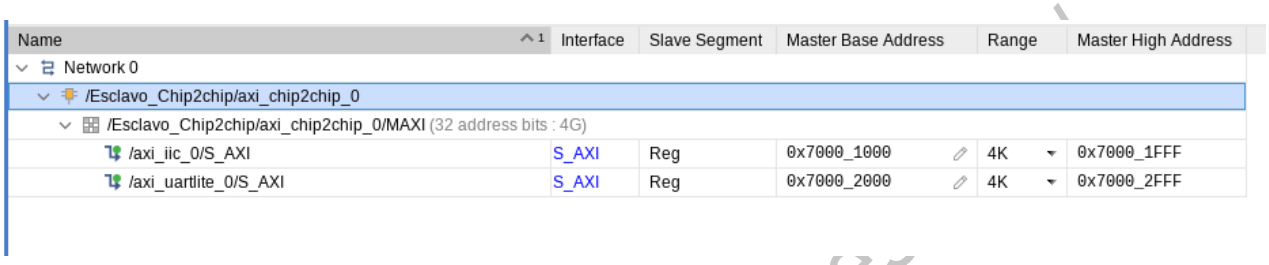
https://soceame.wordpress.com/2024/10/22/como-aumentar-los-pines-de-una-zynq-a-traves-de-una-fpga-expansora/



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/processing_system7_0					
/processing_system7_0/Data (32 address bits : 0x40000000 [1G])					
/axi_gpio_0/S_AXI	S_AXI	Reg	0x4120_0000	4K	0x4120_0FFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x42C0_0000	4K	0x42C0_0FFF
/Master_chip2chip/axi_chip2chip_0/s_axi	s_axi	Mem0	0x7000_0000	512K	0x7007_FFFF

- **Direcciones de la expansora**

Para la expansora se utilizan las direcciones de memoria reservadas por la Zynq sin exceder el espacio reservado.



Name	Interface	Slave Segment	Master Base Address	Range	Master High Address
Network 0					
/Esclavo_Chip2chip/axi_chip2chip_0					
/Esclavo_Chip2chip/axi_chip2chip_0/MAXI (32 address bits : 4G)					
/axi_iic_0/S_AXI	S_AXI	Reg	0x7000_1000	4K	0x7000_1FFF
/axi_uartlite_0/S_AXI	S_AXI	Reg	0x7000_2000	4K	0x7000_2FFF

NOTA: Es importante tener en cuenta en las direcciones de memoria a la hora de irse a Vitis (antiguo Vivado SDK) que la Zynq sólo reconoce sus direcciones de memoria propias, es decir aquellas que están en el proyecto con la Zynq. Por lo que las direcciones de memoria de la expansora no van a estar disponibles en Vitis, porque nadie las va a generar en C. Entonces, tiene que ser el propio usuario quien incorpore las direcciones de memoria a mano en un fichero .h para tenerlas disponibles, y ser cuidadoso de que al añadir más bloques IP se generan nuevas direcciones de memoria.

Con todo esto ya quedarían conectadas la Zynq y la expansora a través del Aurora_8b10b.

Notas finales ejemplo 2

NOTA: Los bloques *AURORA* son bloques que pueden ser a veces complejos y suelen dar fallos complejos de solucionar. Se recomienda manejarlos con cuidado y en caso de que den fallos de forma repetida borrarlos y volverlos a instanciar.

NOTA 2: los fallos más recurrentes con el bloque AURORA son los relojes de entrada que no están bien conectados, que le falte alguna señal de entrada o que alguna señal de salida no esté conectada*(revisar que user_clk_out esté conectado a axi_c2c_phy_clk)*

Bibliografía

- Aurora 8b/10b – PG046 https://docs.amd.com/v/u/8.3-English/pg046-aurora-8b10b
- AXI_chip2chip – PG067 https://docs.amd.com/r/en-US/pg067-axi-chip2chip
- Ejemplo conceptual XAPP1216 https://docs.amd.com/v/u/en-US/xapp1216-axi-chip2chip-aurora