

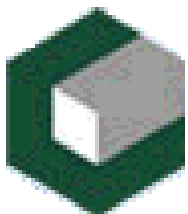
Realizaciones híbridas HW/SW con MicroBlaze

Maestría en Sistemas Digitales

Alejandro J. Cabrera Sarmiento

Dpto. de Automática y Computación
Universidad Tecnológica de La Habana “José Antonio Echeverría”
CUJAE

alex@automatica.cujae.edu.cu



Sumario

- Diseño de sistemas híbridos HW/SW.
- Desarrollo de periféricos **PLB** para MicroBlaze.
- Asistente para crear e importar periféricos.
 - Creación de periféricos
 - Integración de la lógica de usuario
 - Importación de periféricos
- Empleo de periféricos de usuario.



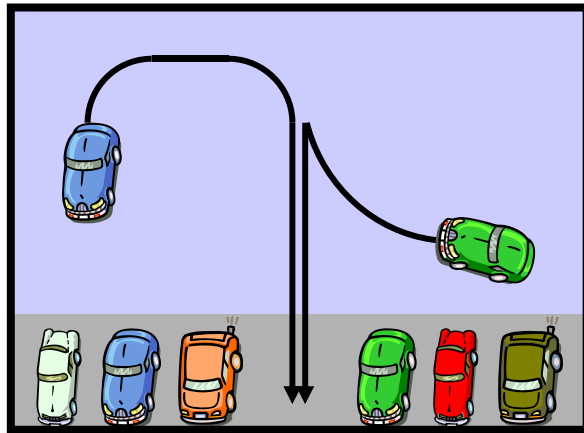
Diseño de sistemas híbridos HW/SW

- Un sistema **híbrido HW/SW** está formado en general por:
 - **Módulos IP del sistema de procesamiento.**
 - **Módulos HW de carácter general** (elementos **disponibles como IP**).
 - **Módulos HW específicos** (**diseñados por el usuario**).
- Con las herramientas de FPGA de Xilinx, la integración entre los diferentes tipos de módulos puede llevarse a cabo con dos estrategias diferentes:
 - Incluyendo el sistema de procesamiento basado en MicroBlaze como **un elemento más en el flujo de diseño de ISE.**
 - **Convertir los módulos HW específicos para que puedan ser usados como periféricos del procesador MicroBlaze en el flujo de diseño de XPS.**
- Ventaja de **reusabilidad.**

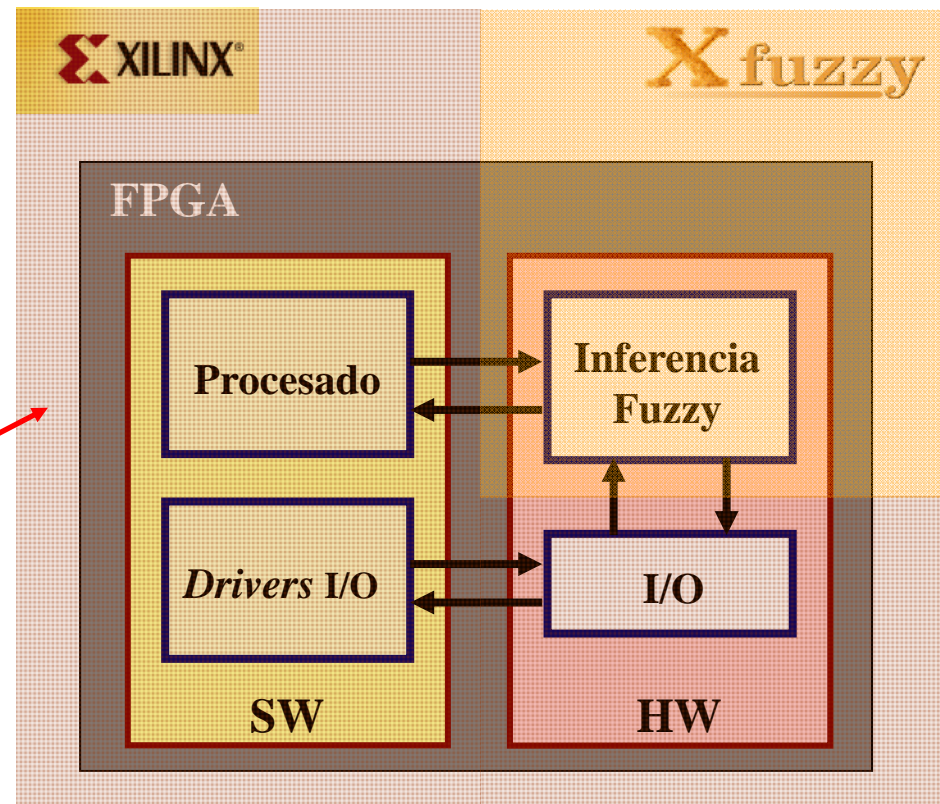
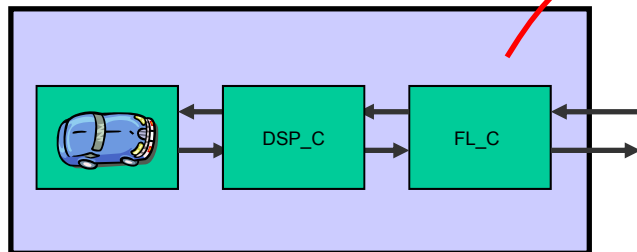


Ejemplo de aplicación

- Controlador Fuzzy de trayectoria del vehículo Romeo4



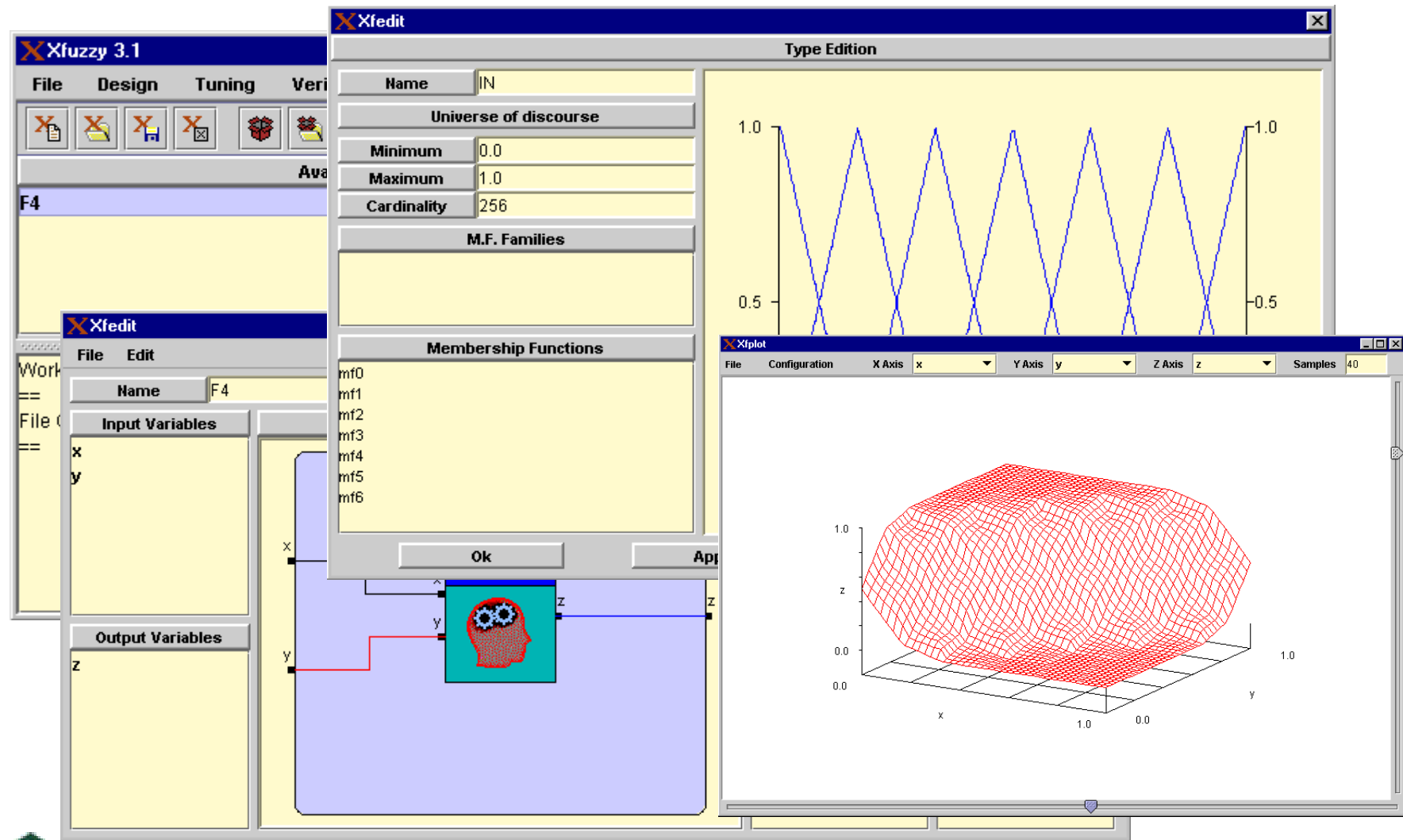
Sistema de control jerárquico



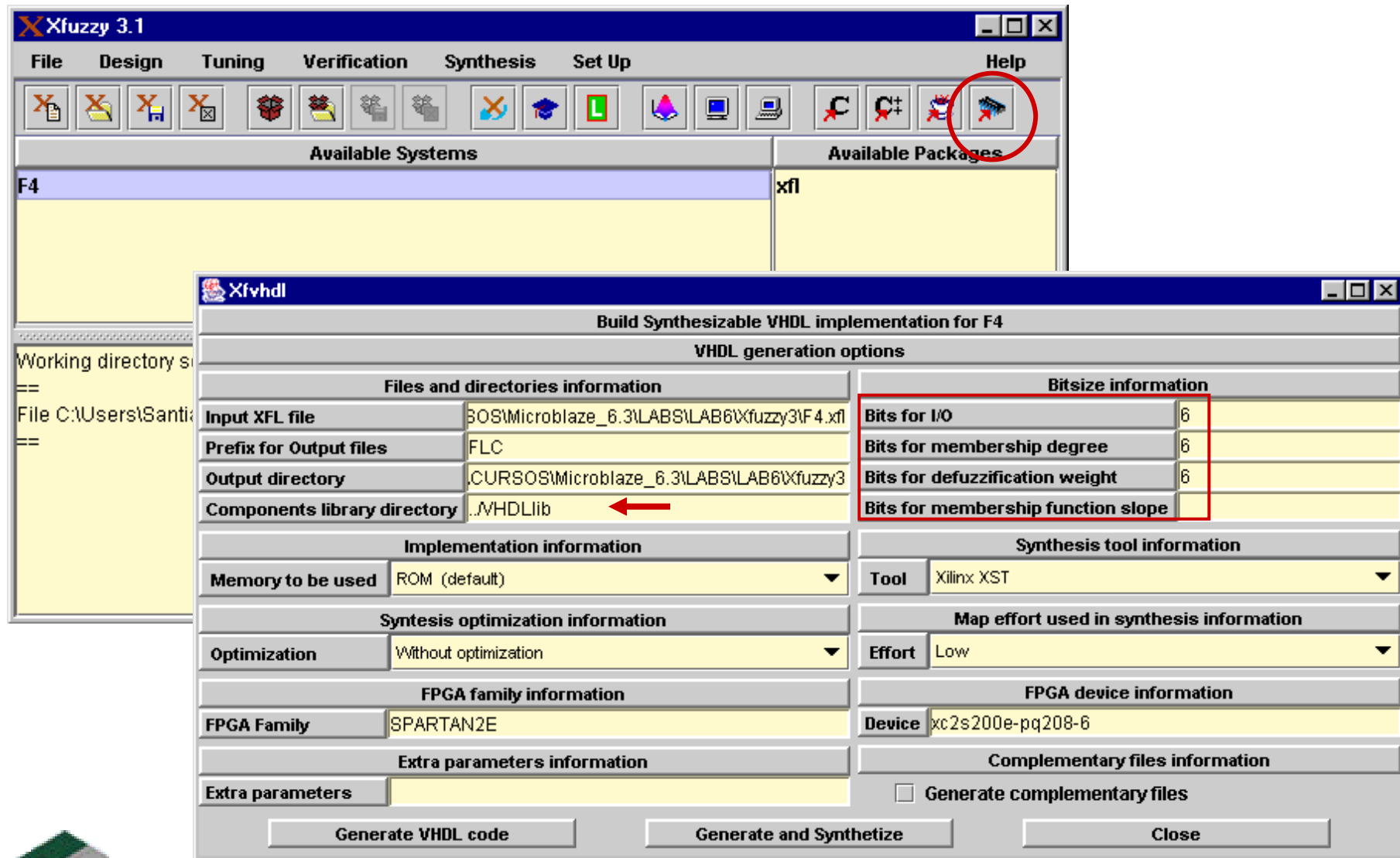
Codiseño Hardware-Software



Desarrollo del módulo de inferencia



Desarrollo del módulo de inferencia (cont.)



Top level del módulo de inferencia

```
library IEEE;  
use IEEE.std_logic_1164.all;
```

```
library FLC6_v1_00_a;  
use FLC6_v1_00_a.Constants.all;  
use FLC6_v1_00_a.Entities.all;
```

Esta entidad habrá
que “conectarla” con
la interfaz para el bus

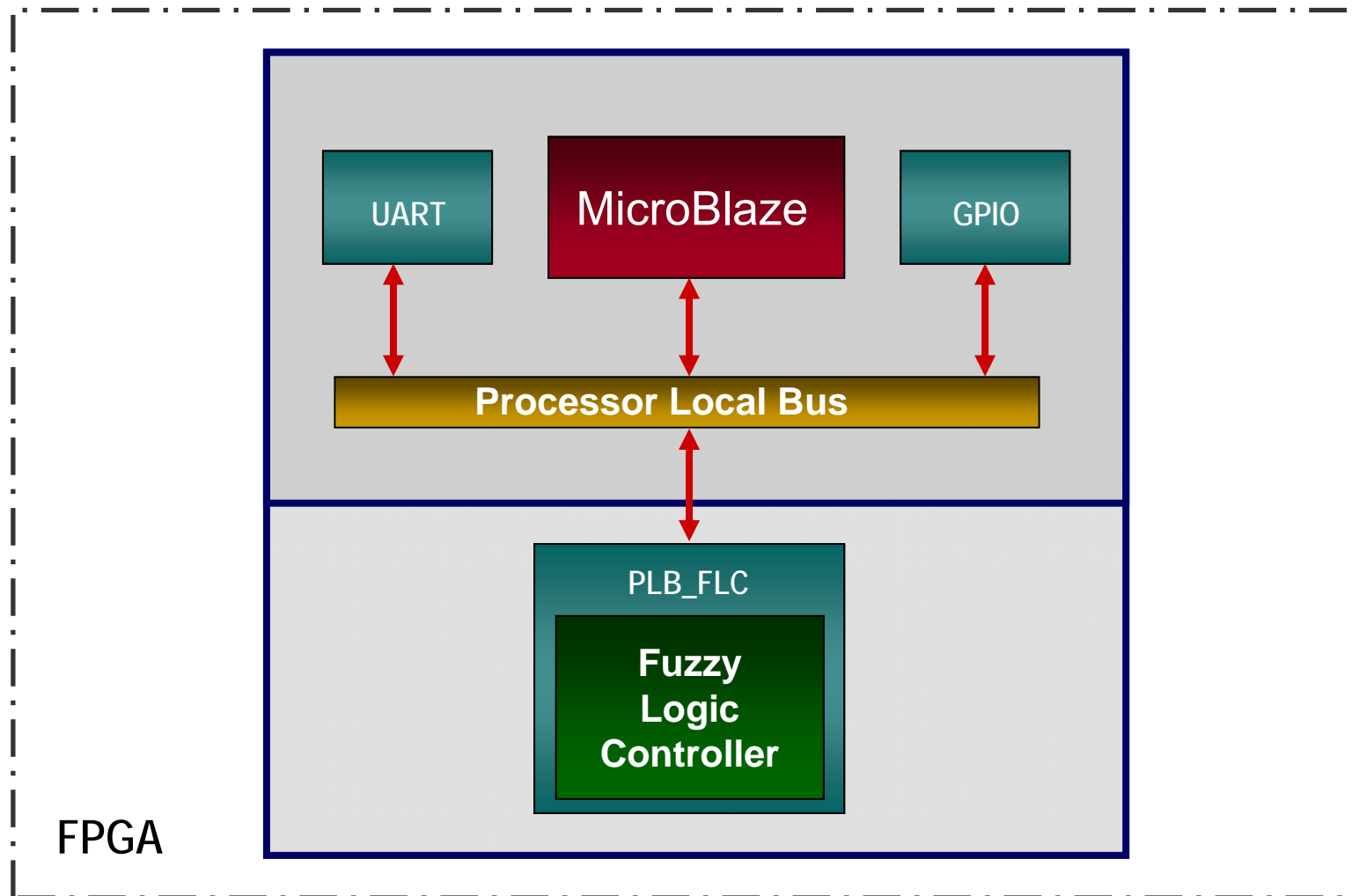
```
-----  
--                               Entity description                               --  
-----
```

```
entity FLC is
```

```
    port(  
        clk          : in std_logic;           -- Clock signal.  
        Reset        : in std_logic;           -- Reset signal.  
        in1           : in std_logic_vector(N downto 1); -- Input 1 signal.  
        in2           : in std_logic_vector(N downto 1); -- Input 2 signal.  
        Output        : out std_logic_vector(N downto 1); -- Output signal.  
        valid_out      : out std_logic;         -- Valid output signal.  
        valid_in       : out std_logic;         -- Valid input signal.
```

```
end FLC;
```

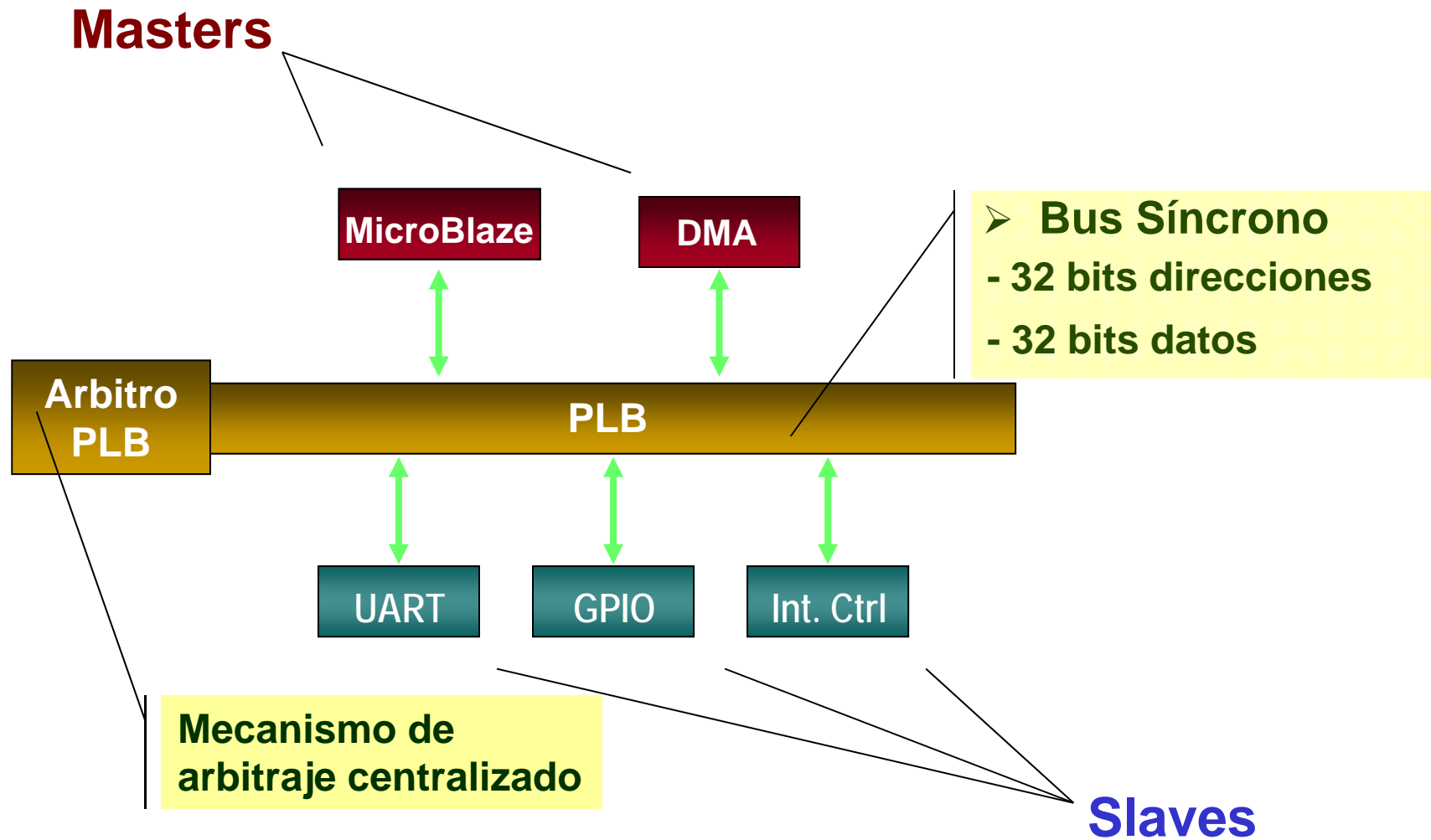
Diseño como periférico de MicroBlaze



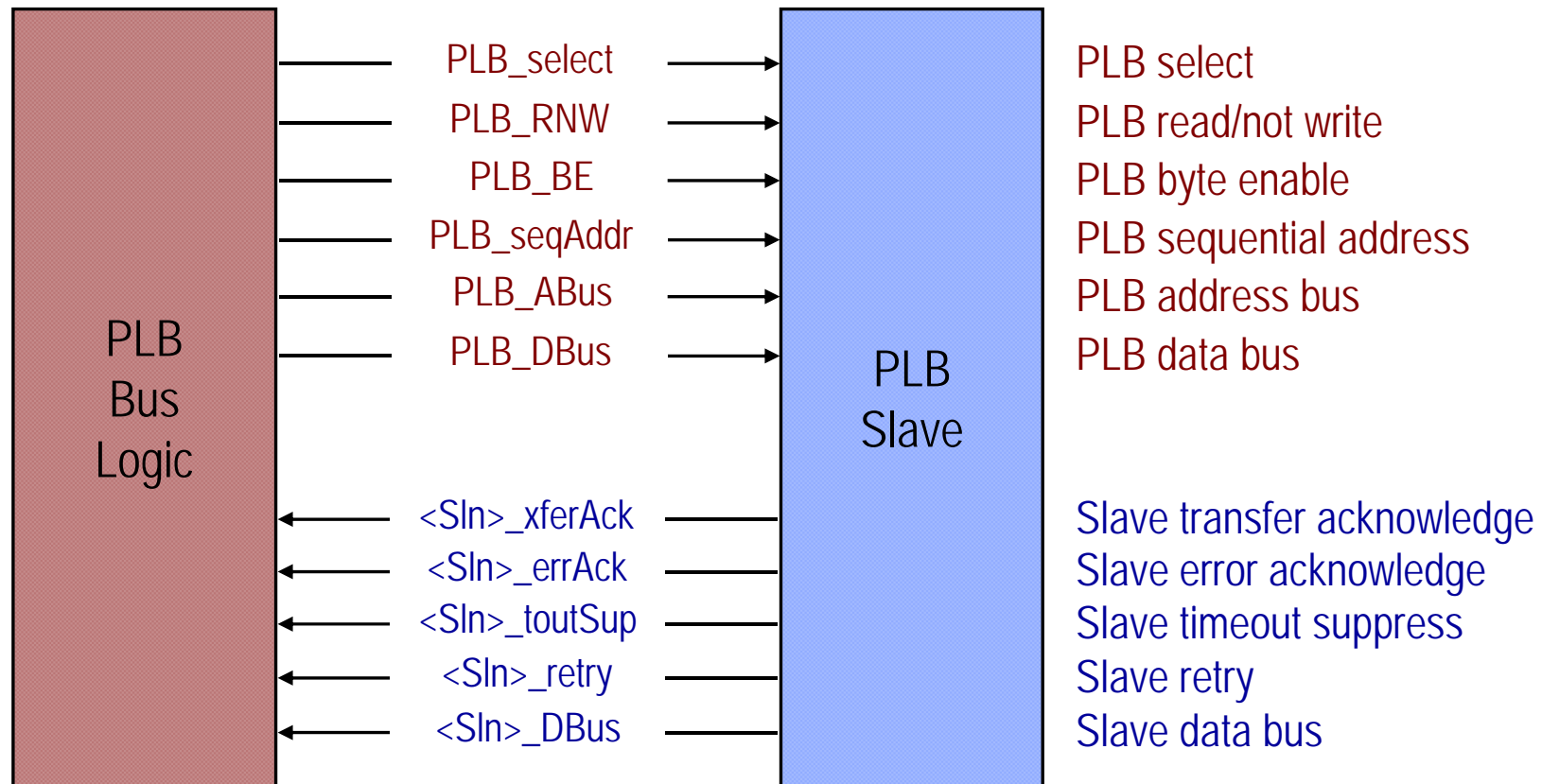
Desarrollo de periféricos para MicroBlaze

- La conexión de periféricos a un procesador se realiza mediante **buses** que agrupan las diferentes líneas de direcciones, datos y control
- Xilinx ha implementado distintas **arquitecturas de interconexión** basadas en el estándar **CoreConnect™** de **IBM**
- El bus **PLB** (*Processor Local Bus*) facilita la conexión de periféricos en un sistema basado en **MicroBlaze** o **PowerPC**.
 - *Custom IPs*
- Cualquier periférico desarrollado, conectable al bus **PLB**, debe cumplir:
 - Las especificaciones del protocolo **PLB**
 - Los requisitos de las herramientas de **XPS**

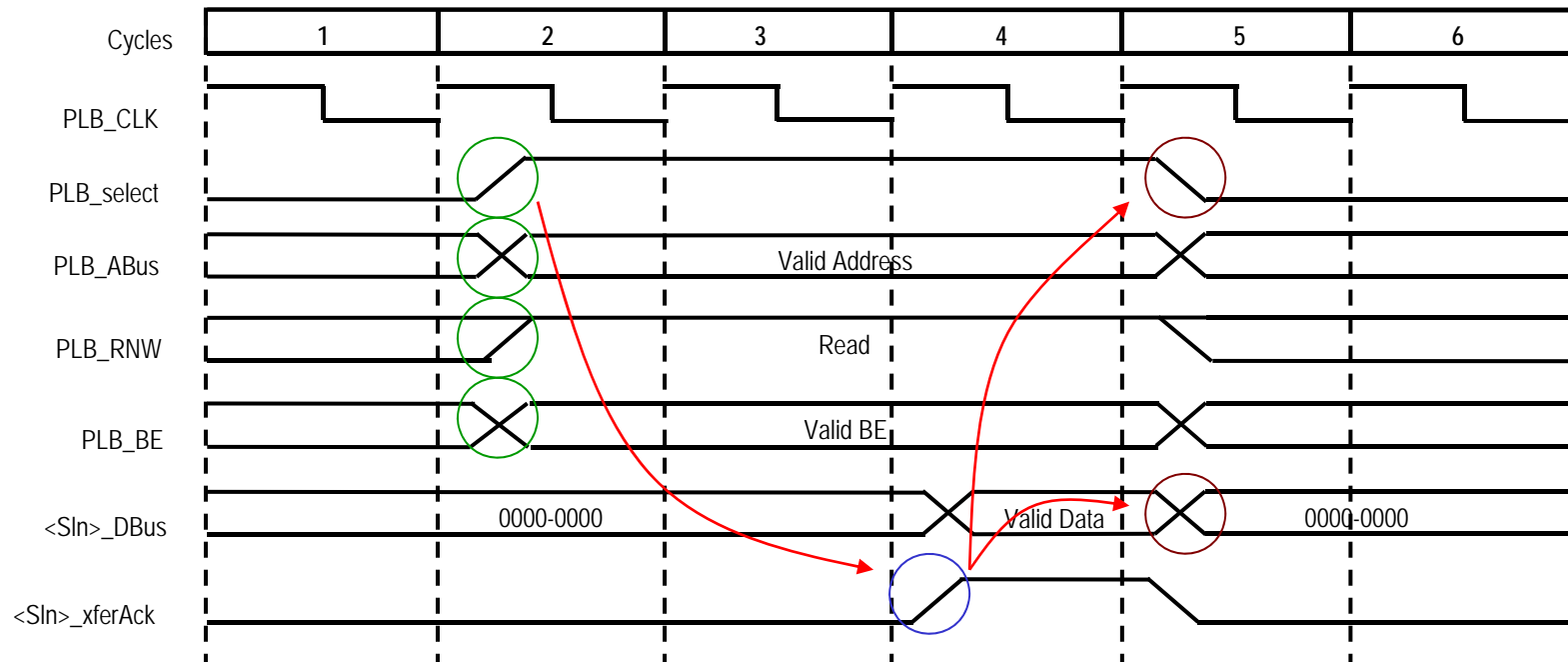
Características del bus *PLB*



Señales de interfaz

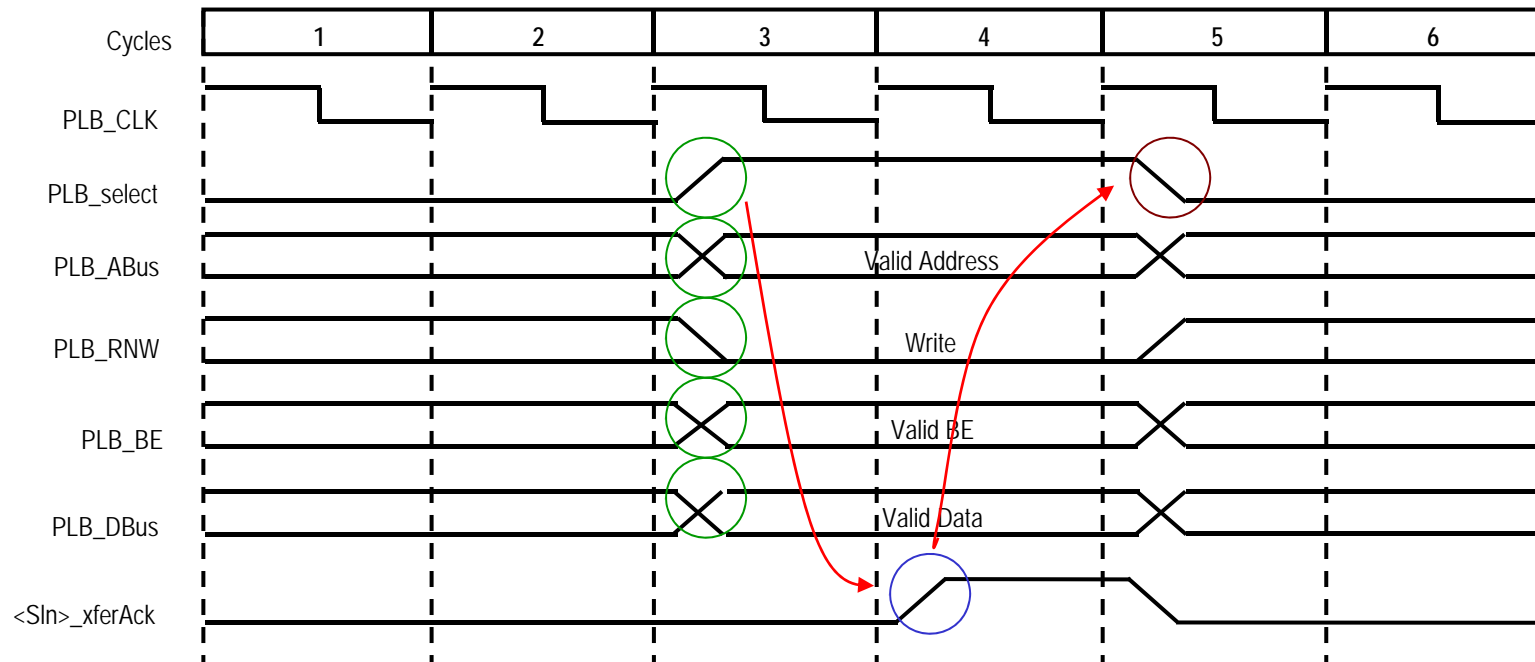


Ciclo de lectura



- 1.- El master de PLB activa la señal **PLB_select** y fija **PLB_ABUS**, **PLB_BE** y **PLB_RNW**
- 2.- El slave activa la señal **PLB_xferAck** que causa que el master lea el **bus de datos** y desactive la señal **PLB_select**

Ciclo de escritura

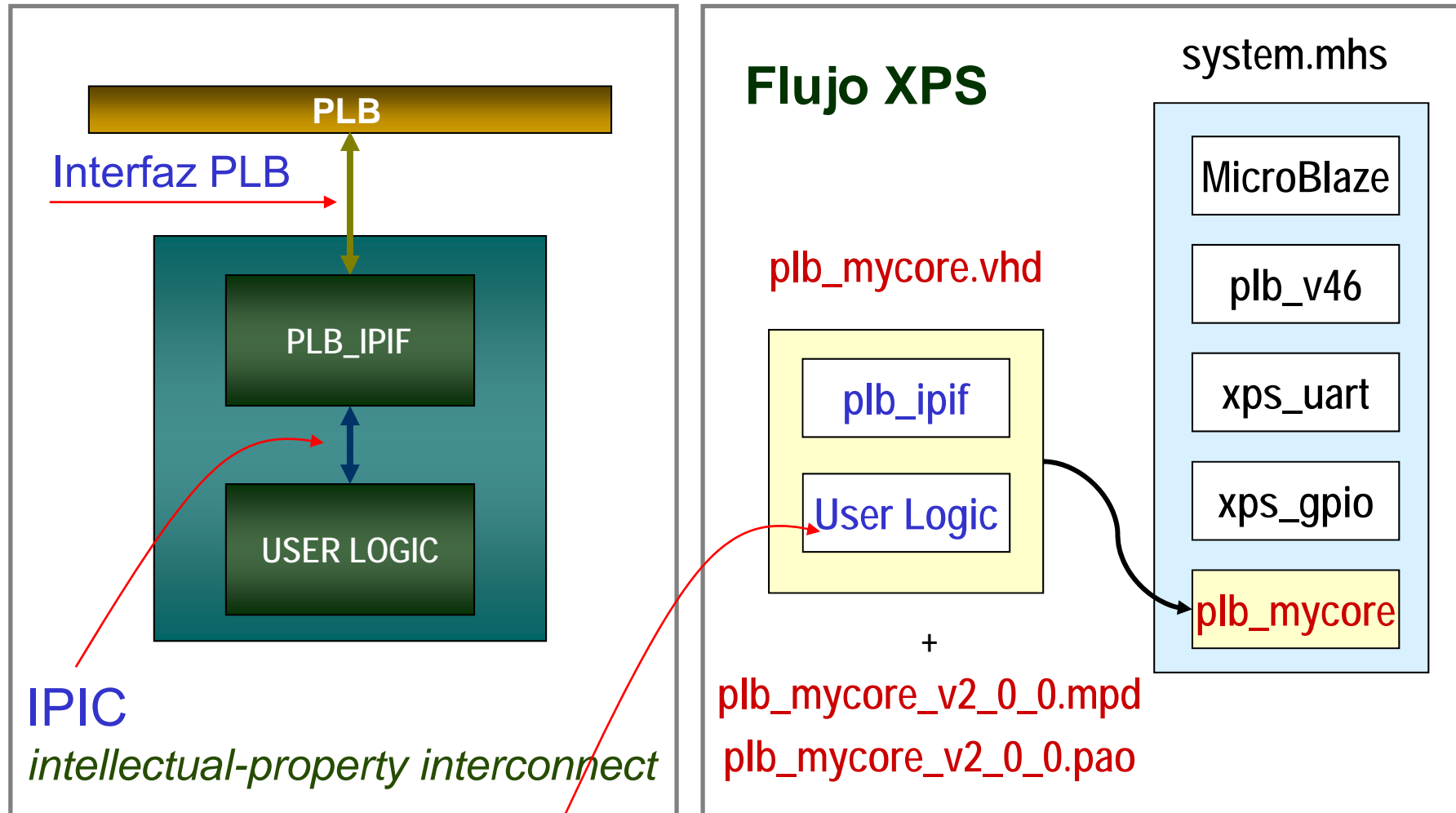


- 1.- El master de PLB activa la señal **PLB_select** y fija **PLB_ABUS**, **PLB_BE**, **PLB_RNW** y **PLB_DBUS**
- 2.- El slave captura el **bus de datos** y activa **PLB_xferAck** que causa que el master desactive **PLB_select**

Plantillas para módulos HW de usuario

- Xilinx proporciona una serie de plantillas (*templates*) que facilitan la conexión de periféricos al bus PLB (o FSL).
 - Las conexiones (y plantillas) a FSL son diferentes
- Estas plantillas consisten en código VHDL que incluye dos componentes:
 - IPIF (*Intellectual-property interface*): interfaz con el *bus PLB*
 - User_logic: interfaz con el HW desarrollado *por el usuario*
- Existen diferentes tipos de plantillas dependiendo del modo de operación (master/slave) del periférico y de los “servicios” proporcionados por la interfaz
- Las plantillas están localizadas en:
 - EDK \ hw \ XilinxProcessorIPLib \ pcores

Plantillas para módulos HW de usuario (cont.)



**Aquí se integra el
HW de usuario**

Create/Import Peripheral Wizard

- Facilita el desarrollo de **módulos IP** para el entorno **EDK**
- Ayuda a **configurar** la funcionalidad de la interfaz IP
- Evita tener que conocer en profundidad la especificación del bus **PLB**.
- Proporciona **ejemplos** de código VHDL para distintas tareas



Create/Import Peripheral Wizard

The screenshot shows the Xilinx Platform Studio interface. The 'Hardware' menu is open, and the 'Create or Import Peripheral...' option is highlighted with a green box. A red arrow points to this option from the left sidebar. The 'IP Catalog' pane on the right lists various IP blocks and their versions. The 'Console' pane at the bottom shows the output of a make command.

Hardware Menu Options:

- Generate Netlist
- Generate Bitstream
- Create or Import Peripheral...**
- Configure Coprocessor...
- Launch Clock Wizard...
- Check and View Core Licenses...
- Clean Netlist
- Clean Bits
- Clean Hardware

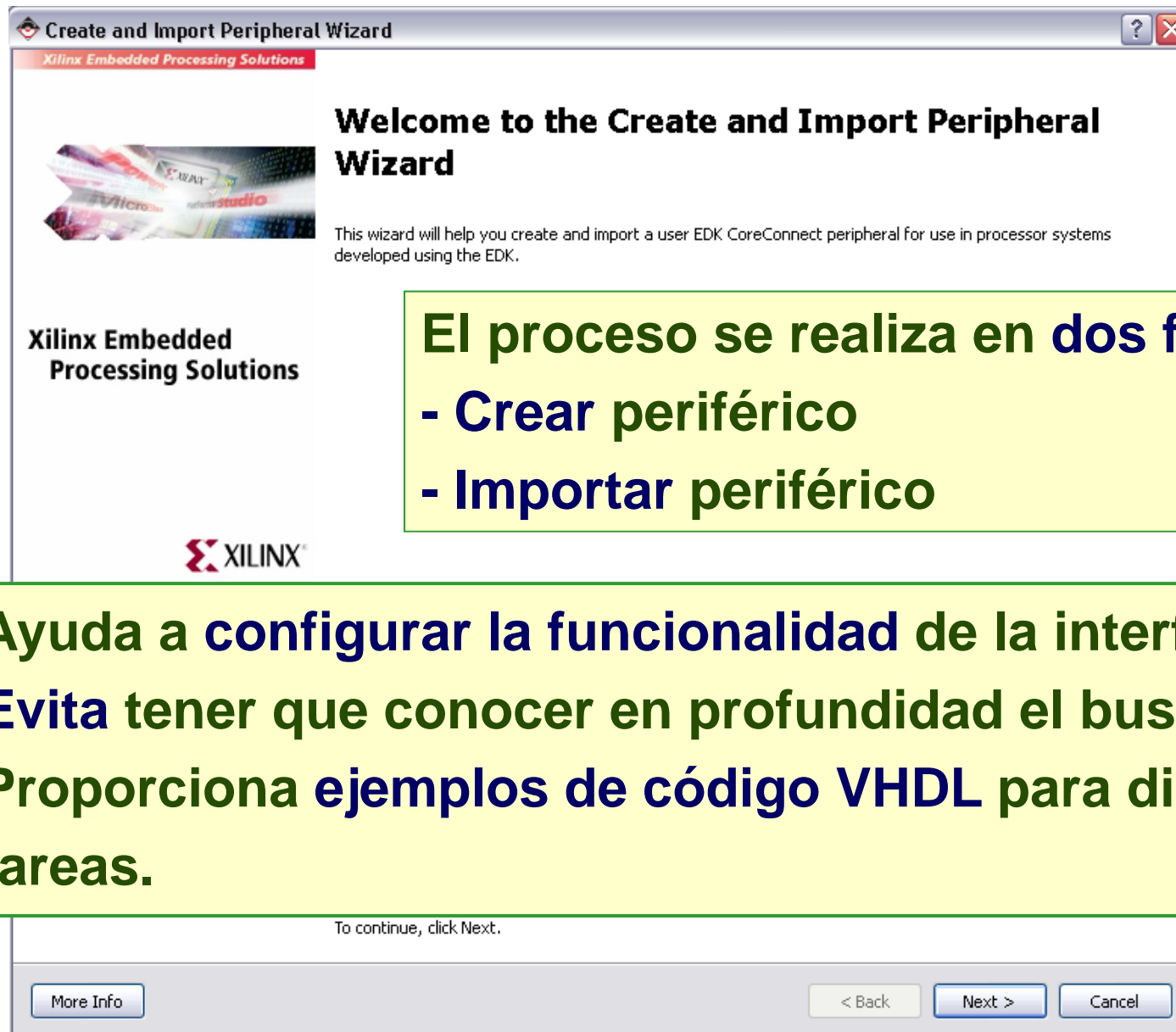
IP Catalog Table:

IP Type	IP Version
lmb_v10	1.00.a
lmb_v10	1.00.a
plb_v46	1.05.a
microblaze	8.00.b
bram_block	1.00.a
lmb_bram_if...	2.10.b
lmb_bram_if...	2.10.b
mdm	2.00.a
xps_intc	2.01.a
xps_gpio	2.00.a
xps_gpio	2.00.a
xps_gpio	2.00.a
xps_timer	1.02.a
xps_uartlite	1.01.a
xps_uartlite	1.01.a
clock_gener...	4.01.a
proc_sys_re...	3.00.a

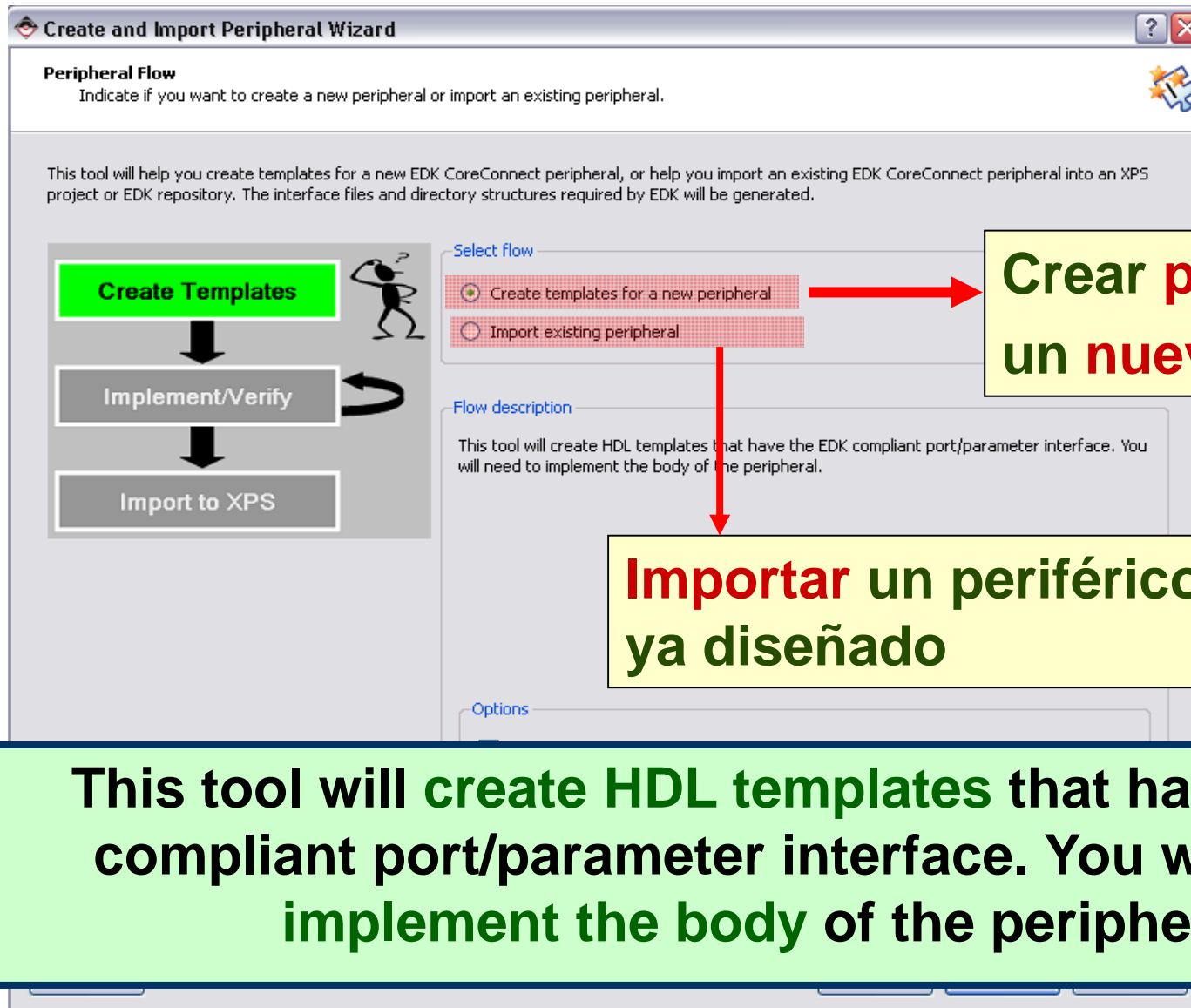
Console Output:

```
make -f system.make init_bram started...
make: Nothing to be done for `init_bram'.
Done!
```

Create/Import Peripheral Wizard (cont.)

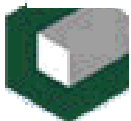


Selección de flujo de diseño



Pasos para la creación de un IP

- **Nombre del módulo** y destino (proyecto o repositorio)
- **Tipo de bus** al que se conectará el periférico
- **Selección y configuración de servicios IPIF**
 - *Intellectual-Property interface*
- **Implementación de la lógica de usuario**
- **Asignación de puertos y definición de parámetros**
- Una vez completados estos pasos el módulo IP puede utilizarse como cualquier otro periférico en EDK



Directorio de Destino

Create Peripheral

Repository or Project
Indicate where you want to store the new peripheral.

A new peripheral can be stored in an EDK repository, or in an XPS project. When stored in an EDK repository, the peripheral can be accessed by multiple XPS projects.

☐ To an EDK user repository (Any directory outside of your EDK installation path)

Repository: **Repositorio**

☒ To an XPS project

Project: **Proyecto XPS**

Peripheral will be placed under:

Repositorio: <Repository-Dir> / MyProcessorIPLib / **pcores**
Proyecto: <XPS-Project-Directory> / **pcores**

Nombre del módulo IP

Create Peripheral

Name and Version
Indicate the name and version of your peripheral.

Enter the name of the peripheral (upper case characters are not allowed). This name will be used to create the peripheral module.

Name: plbflc

Version: 1.00.a

Major revision: 1 Minor revision: 00 Hardware/Software compatibility revision: a

Description:

Nombre (minúsculas)

Versión

Descripción del periférico

Logical library name: plbflc_v1_00_a

All **HDL files** (either created by you or generated by this tool) that are used to implement the peripheral **must be compiled into the logical library name above**. Any other referred logical library in your HDL are assumed to be available in the XPS project where this peripheral is used, or in EDK repositories indicated in the XPS project settings.

Tipo de Bus

Create Peripheral

Bus Interface
Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

- ☒ Processor Local Bus (PLB v4.6)
- ☐ Fast Simplex Link (FSL)

- **Bus PLB**
- **Interfaz FSL**

ATTENTION

Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) bus PLB v4.6 interconnect and the FSL interface.

NOTE - Select the bus interface above and the corresponding link(s) will appear below for that interface.

[CoreConnect Specification](#)

[PLB \(v4.6\) Slave IPIF Specification for single data beat transfer](#)

[PLB \(v4.6\) Slave IPIF Specification for burst data transfer](#)

[PLB \(v4.6\) Master IPIF Specification for single data beat transfer](#)

[PLB \(v4.6\) Master IPIF Specification for burst data transfer](#)

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

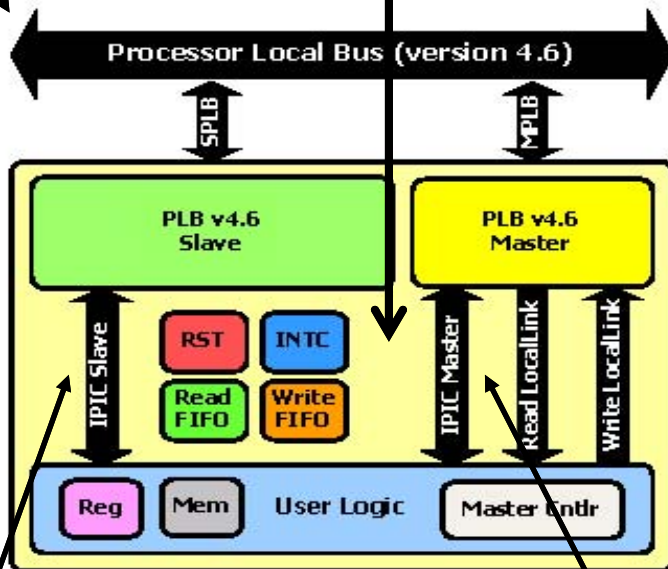
Selección de servicios *IPIF*

Módulo PLB_IPIF

Intellectual-property interface

Interfaz
PLB

will be connected to the PLB (v4.6) interconnect through corresponding PLB IP Interface (IPIF) modules, which provide you with a quick way to implement the interface between the PLB interconnect and the user logic. Besides the standard functions like address decoding provided by the slave IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the implementation of the design.



Slave service and configuration

Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (PLB slave interface will always be included).

☐ Software reset ☒ User logic software register

SERVICIOS:

- **Reset por software**
- **FIFOs de Lectura/Escritura**
- **Soporte de interrupciones**
- **Registros direccionables**
- **Operación como maestro**
- **Espacios de memoria**

IPIC (Intellectual-Property Interconnect)

More Info

< Back

Next >

Cancel

Configuración de servicios *IPIF*

Dependen de los servicios seleccionados

Create Peripheral

Slave Interface
Configure the slave interface of your peripheral

The IPIF slave library provides a quick way to implement a slave interface that supports decoding over various ranges as configured by the user and implements the protocol and timing translation between the PLB v4.6 interconnect and the IPIC (IP InterConnect . interface between user logic and IPIF).

Slave performance

Slave peripherals support single beat read/write data transfers by default. If performance is key to the slave peripheral (i.e. memory controllers), you can have the burst transfer support turned on - this feature provides higher data transfer rates for the PLB Cacheline access and enables the transfer protocol for PLB Fixed Length Burst operations.

☒ Burst and cache-line support

Data width

The native bit width of the internal data bus may be less than or equal to the PLB slave interface data bus width (it is always 32-bit for non-burst slaves and can be 32, 64, or 128-bit for slaves supporting burst). To conserve FPGA resources, set the value to be the same as the smallest PLB master in the system that may interact with your peripheral.

Native data width: 32 bit

32
64
128

Write data buffer

To increase maximum frequency, the IPIF slave library utilizes a write buffer to allow back-to-back data beat transfers on the PLB bus even when the user logic is not capable of acknowledging the writes this quickly. A small buffer will cause flow control on the PLB v4.6 interconnect, while a large buffer takes up FPGA resources. Setting the depth to 0 will eliminate the write data buffer (some peripherals do not need a write buffer or either the user logic will instantiate a buffer alternatively), but keep in mind that excluding the write buffer may expose the user logic to critical path.

Write buffer depth: 16

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

Configuración de servicios *IPIF* (cont.)

Create Peripheral

FIFO Service

Configure the read/write FIFO in the IPIF.

Read/Write FIFO provides data buffering service, which may automatically utilizes SRL16 primitives for the memory medium instead of BRAM primitives if packet mode is turned off and only 4 to 16 words of FIFO memory depth is needed.

☒ Include Read FIFO

☒ Include Write FIFO

☒ Use packet mode

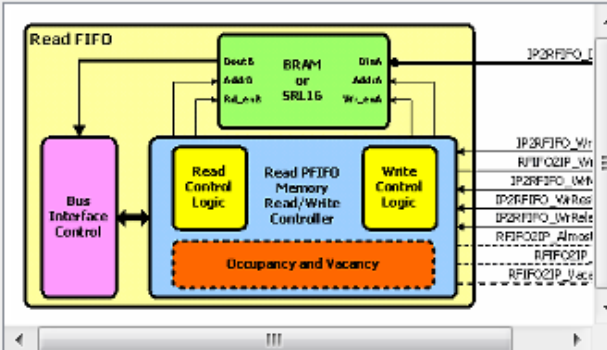
☒ Use vacancy calculation

Number of Read FIFO entries: 128

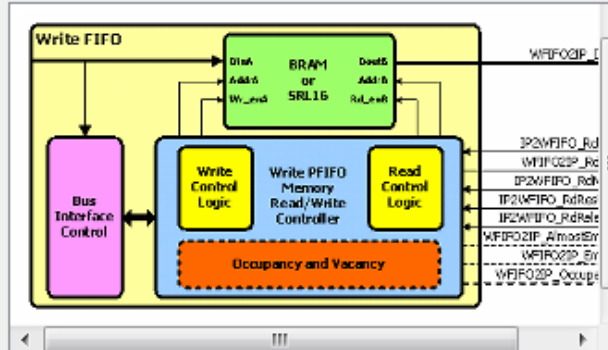
☒ Use packet mode

☒ Use vacancy calculation

Number of Write FIFO entries: 512



Datasheet



Datasheet

[More Info](#)[< Back](#)[Next >](#)[Cancel](#)

Configuración de servicios *IPIF* (cont.)

Interrupt Service

Configure interrupt handling.

The interrupt control service provides interrupt capture support which captures and coalesces various interrupts generated from IPIF, other design blocks and user logic into a single interrupt output.

[Datasheet](#)

Configuración de IT

Device ISC

Device ISC (Interrupt Source Controller) coalesces all captured internal interrupts into a single output signal. You may eliminate Device ISC if all interrupts come from the user logic.

☒ Use Device ISC (interrupt source controller)

Priority Encoder

Device ISC Priority Encoder (Interrupt ID register) indicates which interrupt source has a pending interrupt. It is useful in aiding the user interrupt service routine to resolve the source of an interrupt.

☒ Use Device ISC Priority Encoder service

User logic interrupt

Number of interrupts generated by user-logic:

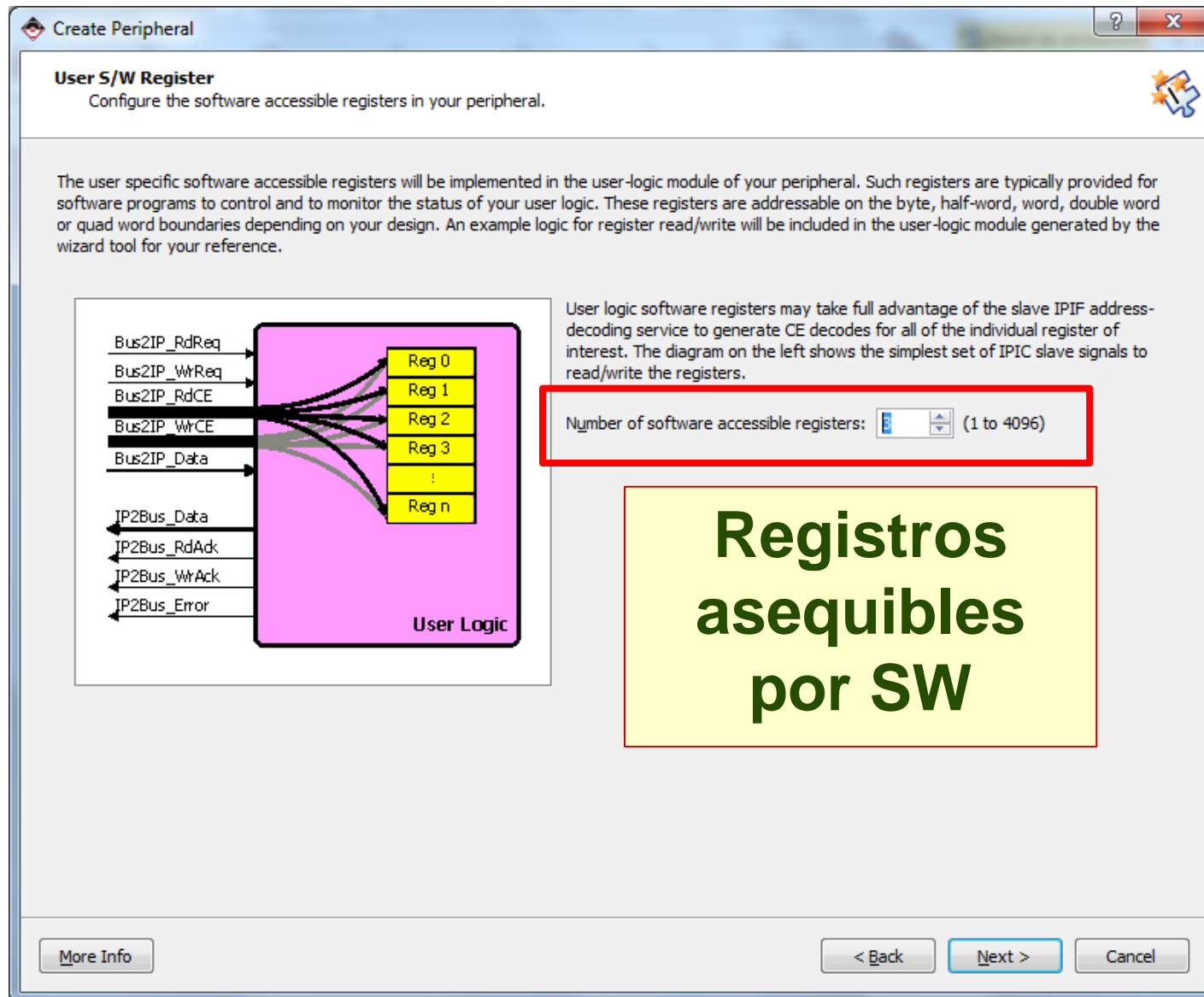
Capture mode:

The input interrupt from the user logic has no additional capture processing applied to it. It is immediately sent to the IP ISC Interrupt Enable gating logic.

[More Info](#)

[< Back](#) [Next >](#) [Cancel](#)

Configuración de servicios *IPIF* (cont.)



Configuración de servicios *IPIF* (cont.)

Create Peripheral

User Memory Space
Indicate the additional memory spaces required by your peripheral.

The user specific memory regions will be implemented in the user-logic module of your peripheral. Such memory spaces are typically seen in memory peripherals like external memory controllers. An example logic inferring multiple Block RAMs for memory read/write will be included in the user-logic module generated by the wizard tool for your reference.

User logic with memory models requires an address range CS decode, a read/write signal and the address bits to access individual memory location. The diagram on the left shows the simplest set of IPIC slave signals to access the memories.

Number of user address ranges:

1
2
3
4
5
6
7
8

Espacios de memoria directamente direccionables x MicroBlaze

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

Configuración *IPIC*

Create Peripheral

IP Interconnect (IPIC)

Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral will be connected to the PLB (v4.6) interconnect through suitable IPIF master/slave module(s). Your custom logic from the user-logic module interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present, some are pre-selected based on the IPIF services you required, and you can choose other optional ports to be included in the design based on your needs.

Note: all IPIC ports are active high.

Peripheral

```
graph TD; subgraph Peripheral; direction TB; Slave[PLB v4.6 Slave]; Others[Other Blocks]; Master[PLB v4.6 Master]; end; UserLogic[User Logic]; Slave <-->|IPIC for slave| UserLogic; Others <-->|IPIC for others| UserLogic; Master <-->|IPIC for master| UserLogic;
```

Bus2IP_Clk

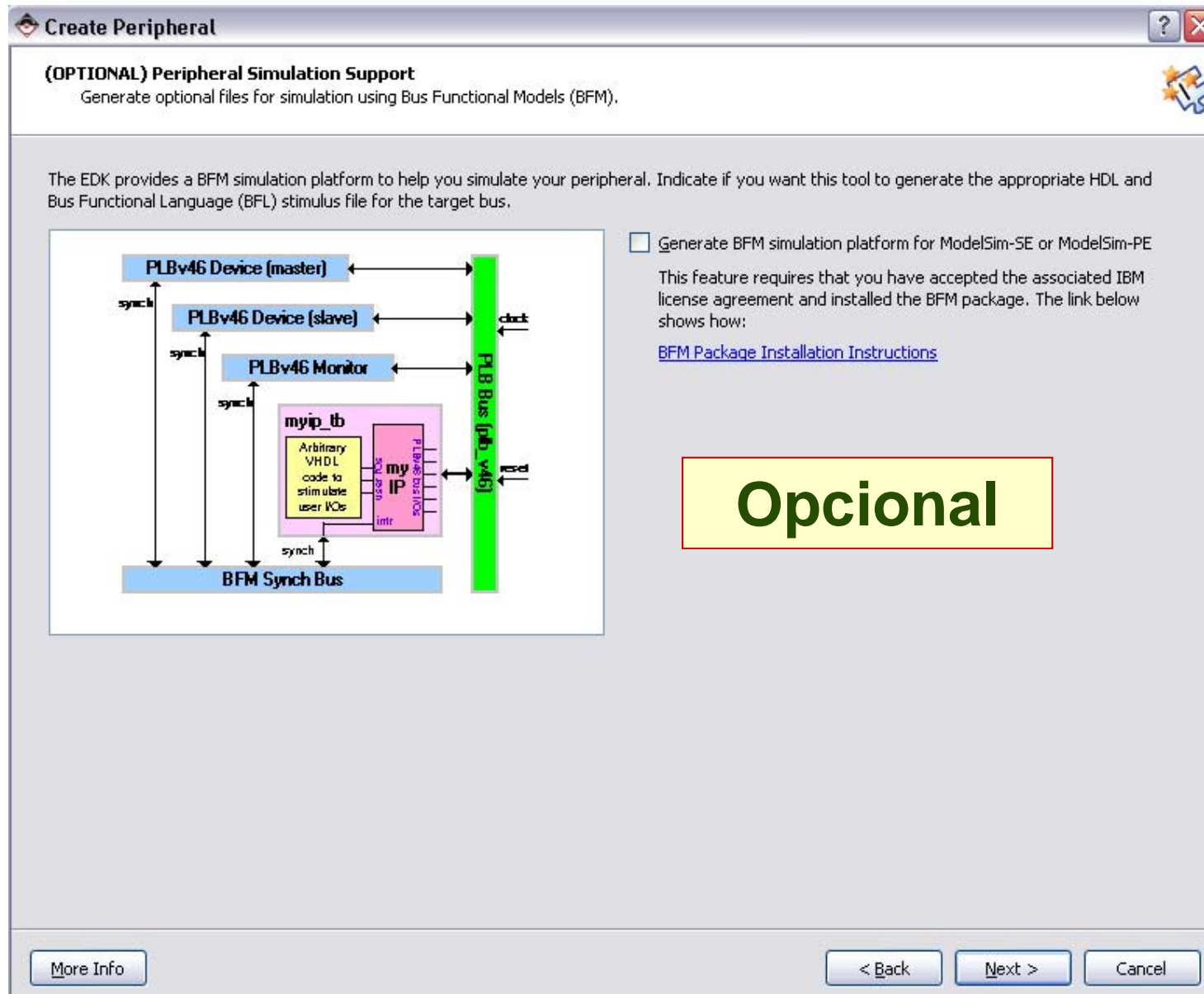
Synchronization clock provided to the user logic. All IPIC signals are synchronous to this clock. It is identical to the input <bus>_Clk signal of the peripheral. No additional buffering is provided on the clock; it is passed through as is.

- ☒ Bus2IP_Clk
- ☒ Bus2IP_Reset
- ☐ Bus2IP_Addr
- ☐ Bus2IP_CS
- ☐ Bus2IP_RNW
- ☒ Bus2IP_Data
- ☒ Bus2IP_BE
- ☒ Bus2IP_RdCE
- ☒ Bus2IP_WrCE
- ☒ IP2Bus_Data
- ☒ IP2Bus_RdAck
- ☒ IP2Bus_WrAck
- ☒ IP2Bus_Error

Usualmente no requiere ser modificado

[More Info](#)[< Back](#)[Next >](#)[Cancel](#)

Soporte para Simulación



Opciones de Implementación

Create Peripheral

(OPTIONAL) Peripheral Implementation Support
Generate optional files for hardware/software implementation

Upon completion, this tool will create synthesizable HDL files that implement the IPIF services you requested. A stub 'user_logic' module will be created. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the synthesizable templates, so that you can hook up the generated peripheral to a processor system.

Peripheral (VHDL)

IPIF (VHDL)

User Logic (VHDL)

Note
Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

- ☐ Generate stub 'user_logic' template in Verilog instead of VHDL
- ☒ Generate ISE and XST project files to help you implement the peripheral using XST flow
- ☒ Generate template driver files to help you implement software interface

☒ **Generar proyecto ISE**

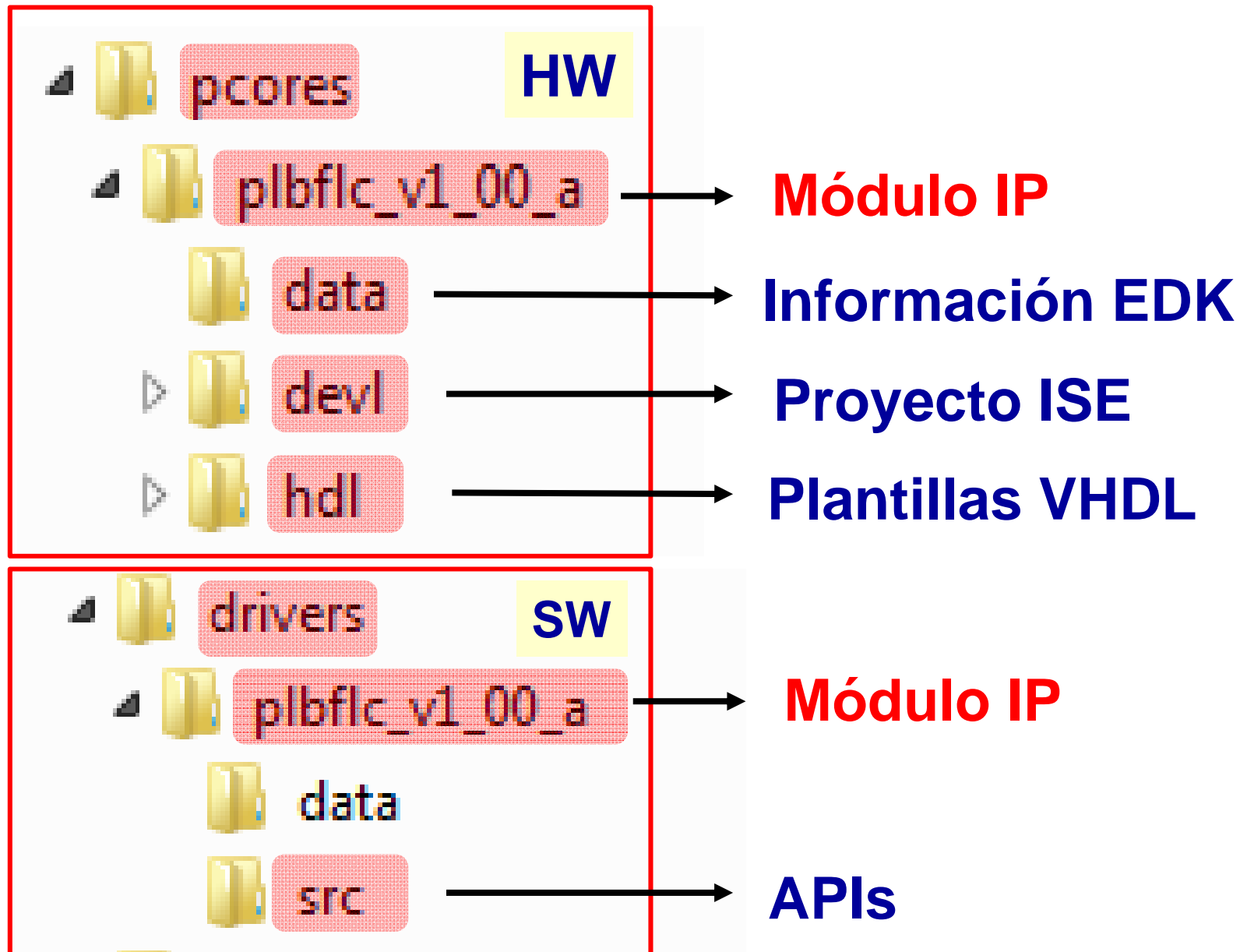
☒ **Generar drivers**

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

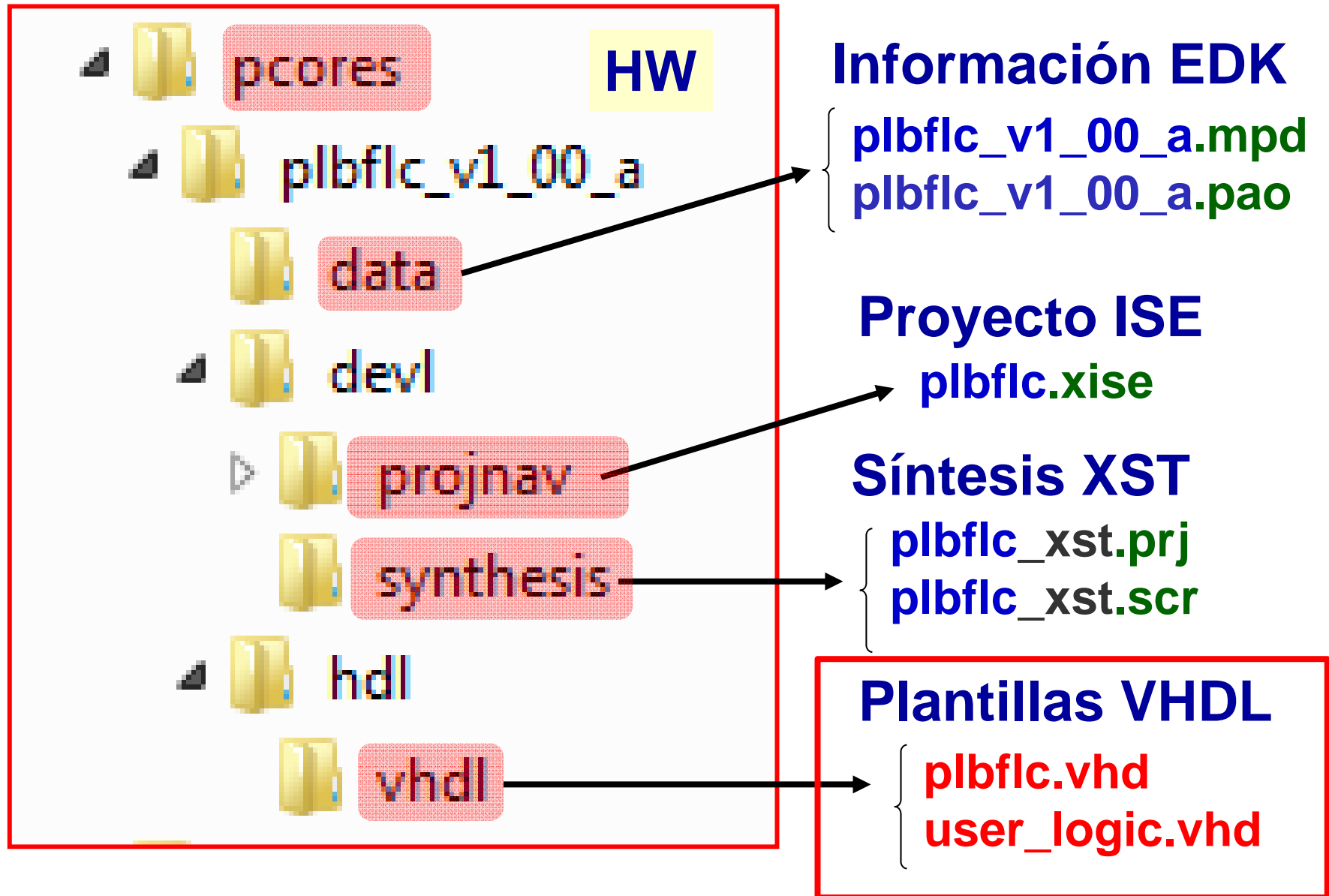
Finalización del proceso de creación



Estructura de Directorios



Estructura de Directorios (cont.)



Fichero *plbflc.vhd* Bibliotecas

```
1  -----
2  -- plbflc.vhd - entity/architecture pair
3  -----
4  -- IMPORTANT:
5  -- DO NOT MODIFY THIS FILE EXCEPT IN THE DESIGNATED SECTIONS.
6  -- SEARCH FOR --USER TO DETERMINE WHERE CHANGES ARE ALLOWED.
7  -- TYPICALLY, THE ONLY ACCEPTABLE CHANGES INVOLVE ADDING NEW
8  -- PORTS AND GENERICS THAT GET PASSED THROUGH TO THE INSTANTIATION
9  -- OF THE USER_LOGIC ENTITY.
10 -----
11 -- Filename:    plbflc.vhd
12 -- Version:     1.00.a
13 -- Description: Top level design, instantiates library components and user logic.
14 -----
15 library ieee;
16 use ieee.std_logic_1164.all;
17 use ieee.std_logic_arith.all;
18 use ieee.std_logic_unsigned.all;
19
20 library proc_common_v3_00_a;
21 use proc_common_v3_00_a.proc_common_pkg.all;
22 use proc_common_v3_00_a.ipif_pkg.all;
23
24 library plbv46_slave_single_v1_01_a;
25 use plbv46_slave_single_v1_01_a.plbv46_slave_single;
26
27 library plbflc_v1_00_a;
28 use plbflc_v1_00_a.user_logic;
```


Fichero *plbflc.vhd* Entidad

```
93  entity plbflc is
94      generic (
95          -- ADD USER GENERICS BELOW THIS LINE -----
96          --USER generics added here
97          -- ADD USER GENERICS ABOVE THIS LINE -----
98
99          -- DO NOT EDIT BELOW THIS LINE -----
100         -- Bus protocol parameters, do not add to or delete
101         C_BASEADDR                : std_logic_vector := X"FFFFFFFF";
102         C_HIGHADDR                 : std_logic_vector := X"00000000";
103         C_SPLB_AWIDTH              : integer          := 32;
104         C_SPLB_DWIDTH              : integer          := 128;
105         C_SPLB_NUM_MASTERS         : integer          := 8;
106         C_SPLB_MID_WIDTH           : integer          := 3;
107         C_SPLB_NATIVE_DWIDTH      : integer          := 32;
108         C_SPLB_P2P                 : integer          := 0;
109         C_SPLB_SUPPORT_BURSTS      : integer          := 0;
110         C_SPLB_SMALLEST_MASTER    : integer          := 32;
111         C_SPLB_CLK_PERIOD_PS       : integer          := 10000;
112         C_INCLUDE_DPHASE_TIMER     : integer          := 0;
113         C_FAMILY                   : string            := "virtex5"
114         -- DO NOT EDIT ABOVE THIS LINE -----
115     );
116     port (
117         -- ADD USER PORTS BELOW THIS LINE -----
118         --USER ports added here
119         -- ADD USER PORTS ABOVE THIS LINE -----
120
121         -- DO NOT EDIT BELOW THIS LINE -----
122         -- Bus protocol ports, do not add to or delete
123         SPLB Clk                    : in  std_logic;
```

Fichero *plbflc.vhd* Arquitectura

```
328 -----
329 -- instantiate User Logic
330 -----
331 USER_LOGIC_I : entity plbflc_v1_00_a.user_logic
332   generic map
333   (
334     -- MAP USER GENERICS BELOW THIS LINE -----
335     --USER generics mapped here
336     -- MAP USER GENERICS ABOVE THIS LINE -----
337
338     C_SLV_DWIDTH      => USER_SLV_DWIDTH,
339     C_NUM_REG         => USER_NUM_REG
340   )
341   port map
342   (
343     -- MAP USER PORTS BELOW THIS LINE -----
344     --USER ports mapped here
345     -- MAP USER PORTS ABOVE THIS LINE -----
346
347     Bus2IP_Clk        => ipif_Bus2IP_Clk,
348     Bus2IP_Reset      => ipif_Bus2IP_Reset,
349     Bus2IP_Data       => ipif_Bus2IP_Data,
350     Bus2IP_BE         => ipif_Bus2IP_BE,
351     Bus2IP_RdCE       => user_Bus2IP_RdCE,
352     Bus2IP_WrCE       => user_Bus2IP_WrCE,
353     IP2Bus_Data       => user_IP2Bus_Data,
354     IP2Bus_RdAck      => user_IP2Bus_RdAck,
355     IP2Bus_WrAck      => user_IP2Bus_WrAck,
356     IP2Bus_Error      => user_IP2Bus_Error
357   );
```


Fichero *user-logic.vhd* Bibliotecas

```
1  -----
2  -- user_logic.vhd - entity/architecture pair
3  -----
4  -- Filename:          user_logic.vhd
5  -- Version:           1.00.a
6  -- Description:       User logic.
7  -- Date:              Sat Nov 21 (by Create and Import Peripheral Wizard)
8  -- VHDL Standard:     VHDL'93
9  -----
10
11 -- DO NOT EDIT BELOW THIS LINE -----
12 library ieee;
13 use ieee.std_logic_1164.all;
14 use ieee.std_logic_arith.all;
15 use ieee.std_logic_unsigned.all;
16
17 library proc_common_v3_00_a;
18 use proc_common_v3_00_a.proc_common_pkg.all;
19
20 -- DO NOT EDIT ABOVE THIS LINE -----
21
22 --USER libraries added here
23
24 -----
```

**Fichero *a modificar*
añadiendo el *HW* de
*usuario***

Fichero *user-logic.vhd* Entidad

```
44 entity user_logic is
45     generic
46         -- ADD USER GENERICS BELOW THIS LINE -----
47         --USER generics added here
48         -- ADD USER GENERICS ABOVE THIS LINE -----
49
50         -- DO NOT EDIT BELOW THIS LINE -----
51         -- Bus protocol parameters, do not add to or delete
52         C_SLV_DWIDTH          : integer          := 32;
53         C_NUM_REG              : integer          := 3
54         -- DO NOT EDIT ABOVE THIS LINE -----
55     );
56     port
57         -- ADD USER PORTS BELOW THIS LINE -----
58         --USER ports added here
59         -- ADD USER PORTS ABOVE THIS LINE -----
60
61         -- DO NOT EDIT BELOW THIS LINE -----
62         -- Bus protocol ports, do not add to or delete
63         Bus2IP_Clk             : in  std_logic;
64         Bus2IP_Reset           : in  std_logic;
65         Bus2IP_Data            : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
66         Bus2IP_BE              : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
67         Bus2IP_RdCE            : in  std_logic_vector(0 to C_NUM_REG-1);
68         Bus2IP_WrCE            : in  std_logic_vector(0 to C_NUM_REG-1);
69         IP2B                    : out std_logic_vector(C_SLV_DWIDTH-1);
70         IP2B                    : out std_logic_vector(C_SLV_DWIDTH-1);
71         IP2B                    : out std_logic_vector(C_SLV_DWIDTH-1);
72         IP2B                    : out std_logic_vector(C_SLV_DWIDTH-1);
73         -- D
74     );
```

Fichero a modificar añadiendo el HW de usuario

Fichero *user-logic.vhd* Arquitectura

```
82  -----
83  -- Architecture section
84  -----
85
86  architecture IMP of user_logic is
87
88      --USER signal declarations added here, as needed for user logic
89
90      -----
91      -- Signals for user logic slave model s/w accessible register example
92      -----
93      signal slv_reg0                : std_logic_vector(0 to C_SLV_DWIDTH-1);
94      signal slv_reg1                : std_logic_vector(0 to C_SLV_DWIDTH-1);
95      signal slv_reg2                : std_logic_vector(0 to C_SLV_DWIDTH-1);
96      signal slv_reg_write_sel       : std_logic_vector(0 to 2);
97      signal slv_reg_read_sel        : std_logic_vector(0 to 2);
98      signal slv_ip2bus_data         : std_logic_vector(0 to C_SLV_DWIDTH-1);
99      signal slv_read_ack             : std_logic;
100     signal slv_write_ack            : std_logic;
101
102     begin
103
104         --USER logic implementation added here
105
106         -----
107         -- Example code to read/write user logic slave model s/w accessible registers
108         --
```

**Declarar la componente
del *top level* y mapearla**

Lectura / Escritura en registros

```
106 -----
107 -- Example code to read/write user logic slave model s/w accessible registers
108 --
109 -- Note:
110 -- The example code presented here is to show you one way of reading/writing
111 -- software accessible registers implemented in the user logic slave model.
112 -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
113 -- to one software accessible register by the top level template. For example,
114 -- if you have four 32 bit software accessible registers in the user logic,
115 -- you are basically operating on the following memory mapped registers:
116 --
117 --      Bus2IP_WrCE/Bus2IP_RdCE      Memory Mapped Register
118 --      "1000"      C_BASEADDR + 0x0
119 --      "0100"      C_BASEADDR + 0x4
120 --      "0010"      C_BASEADDR + 0x8
121 --      "0001"      C_BASEADDR + 0xC
122 --
123 -----
124 slv_reg_write_sel <= Bus2IP_WrCE(0 to 2);
125 slv_reg_read_sel  <= Bus2IP_RdCE(0 to 2);
126 slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2);
127 slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2);
...
```

**Direcciones de
los registros**

Lectura / Escritura en registros (cont.)

```
129  -- implement slave model software accessible register(s)
```

```
130  SLAVE REG WRITE PROC : process( Bus2IP Clk )
```

```
131  begin
```

```
132
```

```
133      if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
```

```
134          if Bus2IP_Reset = '1' then
```

```
135              slv_reg0 <= (others => '0');
```

```
136              slv_reg1 <= (others => '0');
```

```
137              slv_reg2 <= (others => '0');
```

```
138          else
```

```
139              case slv_reg_write_sel is
```

```
14  165  -- implement slave model software accessible register(s) read mux
```

```
14  166  SLAVE REG READ PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1,
```

```
14  167                                slv_reg2 ) is
```

```
14  168  begin
```

```
14  169
```

```
14  170      case slv_reg_read_sel is
```

```
14  171          when "100" => slv_ip2bus_data <= slv_reg0;
```

```
14  172          when "010" => slv_ip2bus_data <= slv_reg1;
```

```
14  173          when "001" => slv_ip2bus_data <= slv_reg2;
```

```
14  174          when others => slv_ip2bus_data <= (others => '0');
```

```
14  175      end case;
```

```
14  176
```

```
177  end process SLAVE_REG_READ_PROC;
```

Escritura

Lectura

Integración de la lógica de usuario

```
151
152      --USER logic implementation added here
153
154      signal output      : std_logic_vector(0 to C_DWIDTH-1);
155      signal valid_out    : std_logic;
156      signal valid_in     : std_logic;
157
158      -- COMPONENTS
159      component FLC is
160      port (clk           : in std_logic;           -- Clock signal.
161           reset          : in std_logic;          -- Reset signal.
162           in1            : in std_logic_vector(N downto 1); -- Input 1 signal.
163           in2            : in std_logic_vector(N downto 1); -- Input 2 signal.
164           output         : out std_logic_vector(N downto 1); -- Output signal.
165           valid_out      : out std_logic;          -- Valid output signal.
166           valid_in       : out std_logic);         -- Valid input signal.
167
168      end component FLC;
169
170      begin
171
172      FIM : FLC port map(
173          clk      => Bus2IP_Clk,
174          reset    => Bus2IP_Reset,
175          in1      => slv_reg0 ,
176          in2      => slv_reg1 ,
177          output   => output,
178          valid_out => valid_out,
179          valid_in  => valid_in
180      );
181
```


Integración de la lógica de usuario

```
FLC port map(  
    clk          => Bus2IP_Clk,  
    reset        => Bus2IP_Reset,  
    in1          => slv_reg0 ,  
    in2          => slv_reg1 ,  
    output       => output,  
    valid_out    => valid_out,  
    valid_in     => valid_in  
);
```

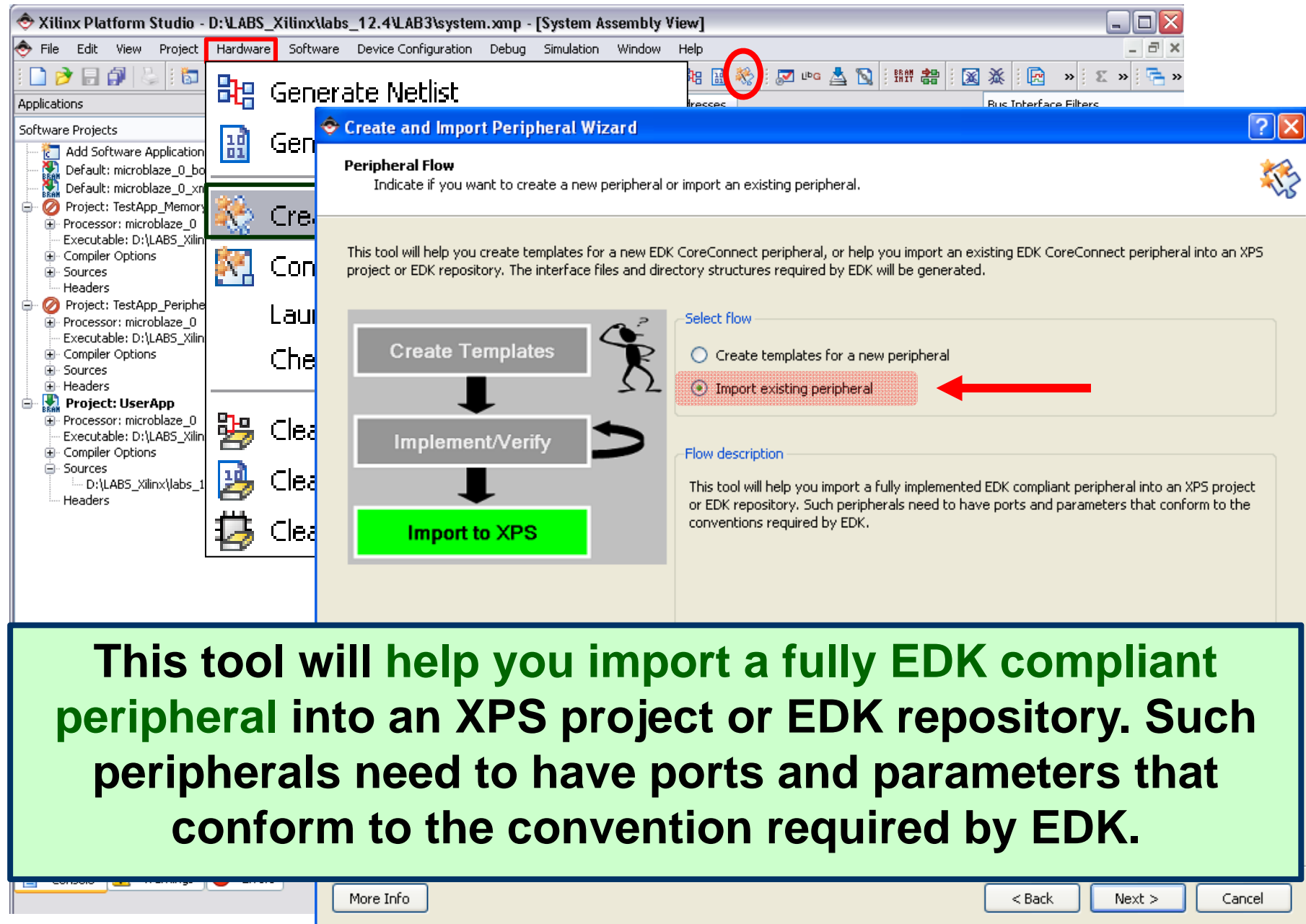
**Conexión del HW de usuario con
los registros del módulo IP**

Importación de Periféricos

- Importa un periférico **ya implementado** a un repositorio o proyecto XPS.
- Suele usarse **después de crear las plantillas** para implementar el módulo IP.
- También puede usarse para importar diseños creados sin plantillas, pero en este caso hay que asegurar que se cumplen las convenciones que requiere EDK.



Importación de periféricos



The screenshot shows the Xilinx Platform Studio interface with the 'Hardware' menu open. The 'Create and Import Peripheral Wizard' is displayed, showing a flow diagram with steps: 'Create Templates', 'Implement/Verify', and 'Import to XPS'. The 'Import existing peripheral' option is selected under 'Select flow'. A red arrow points to this option. The 'Flow description' text states: 'This tool will help you import a fully implemented EDK compliant peripheral into an XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the conventions required by EDK.'

This tool will help you import a fully EDK compliant peripheral into an XPS project or EDK repository. Such peripherals need to have ports and parameters that conform to the convention required by EDK.

Localización, nombre y versión

Create Peripheral

Repository or Project
Indicate where you want to store the peripheral.

A new peripheral can be stored in an EDK user repository or in an XPS project.

☐ To an EDK user repository (Any directory)

Repository:

☒ To an XPS project

Project:

Peripheral will be placed under:

[More Info](#)

Import Peripheral

Name and Version
Indicate the name of your peripheral and if using the EDK peripheral version naming scheme.

Enter name of the top VHDL entity or Verilog module of your peripheral.

Name:

☒ Use version: 1.00.a

Major revision: Minor revision: Hardware/Software compatibility revision:

Nombre

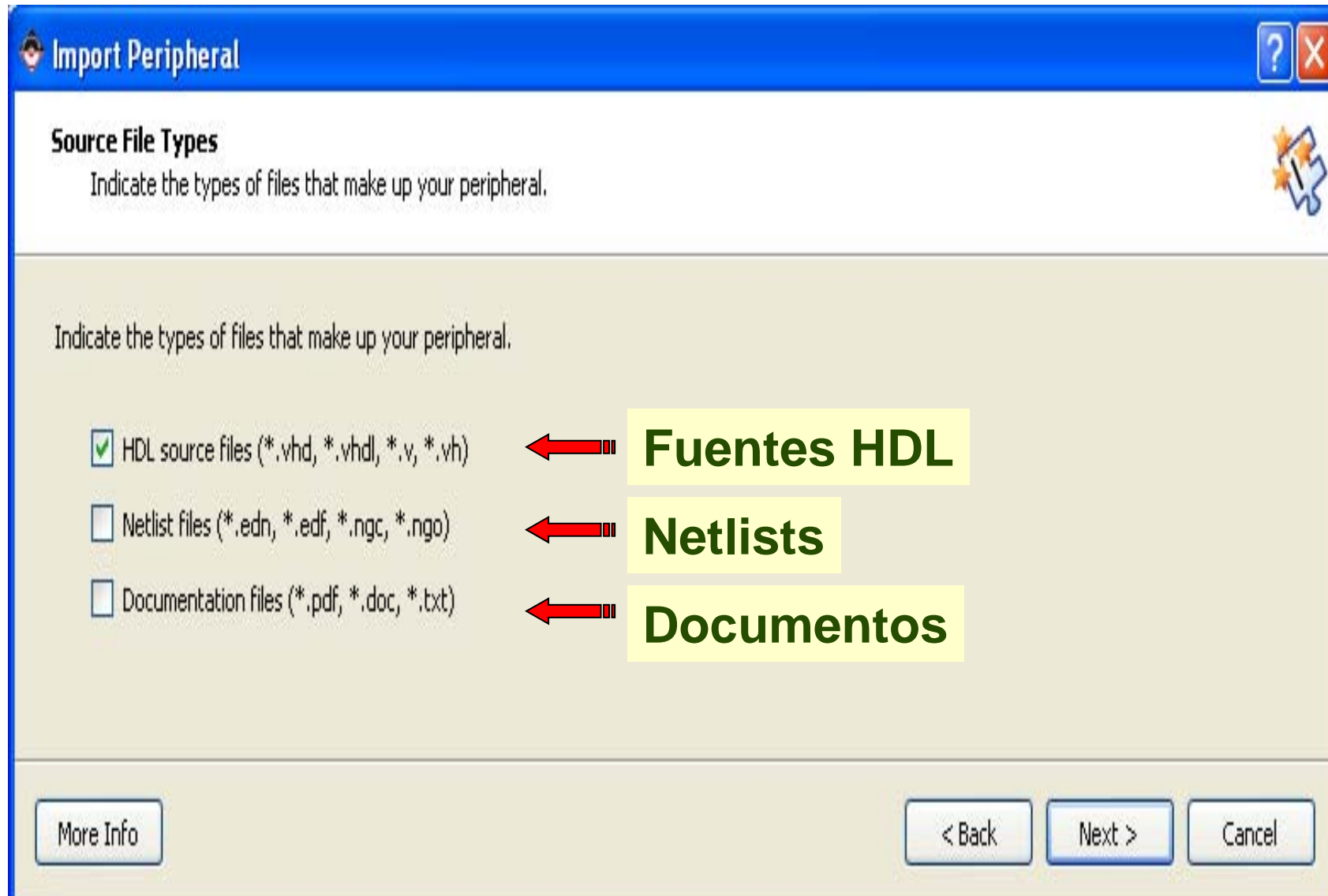
Versión

Logical library name: plbflc_v1_00_a

All the files for this peripheral are compiled into the logical library named above. If the peripheral refers to other logical libraries, they are either assumed to be available in the current project or in the repositories accessible through the current project settings, or will be imported along with the peripheral. Since all design files are compiled in the same directory, using logical libraries other than given above may cause name space conflicts.

[More Info](#) [< Back](#) [Next >](#) [Cancel](#)

Ficheros Fuente



Ficheros Fuente HDL

Import Peripheral

HDL Source Files
Indicate how this tool should locate the HDL files that make up your peripheral.

HDL language used to import peripheral:

☒ Use data (*.mpd) collected during a previous invocation of this tool

☐ Use data (*.mpd) collected during a previous invocation of this tool

C:\Labs_xilinx\LAB3\pcores\plbflc_v1_00_a\data\plbflc_v2_1_0.mpd

How to locate your HDL source files and dependent library files

☐ Use an XST project file (*.prj)
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines

☒ Use existing Peripheral Analysis Order file (*.pao)
C:\Labs_xilinx\LAB3\pcores\plbflc_v1_00_a\data\plbflc_v2_1_0.pao

☐ Browse to your HDL source and dependent library files (*.vhd, *.vhdl, *.v, *.vh) in next step

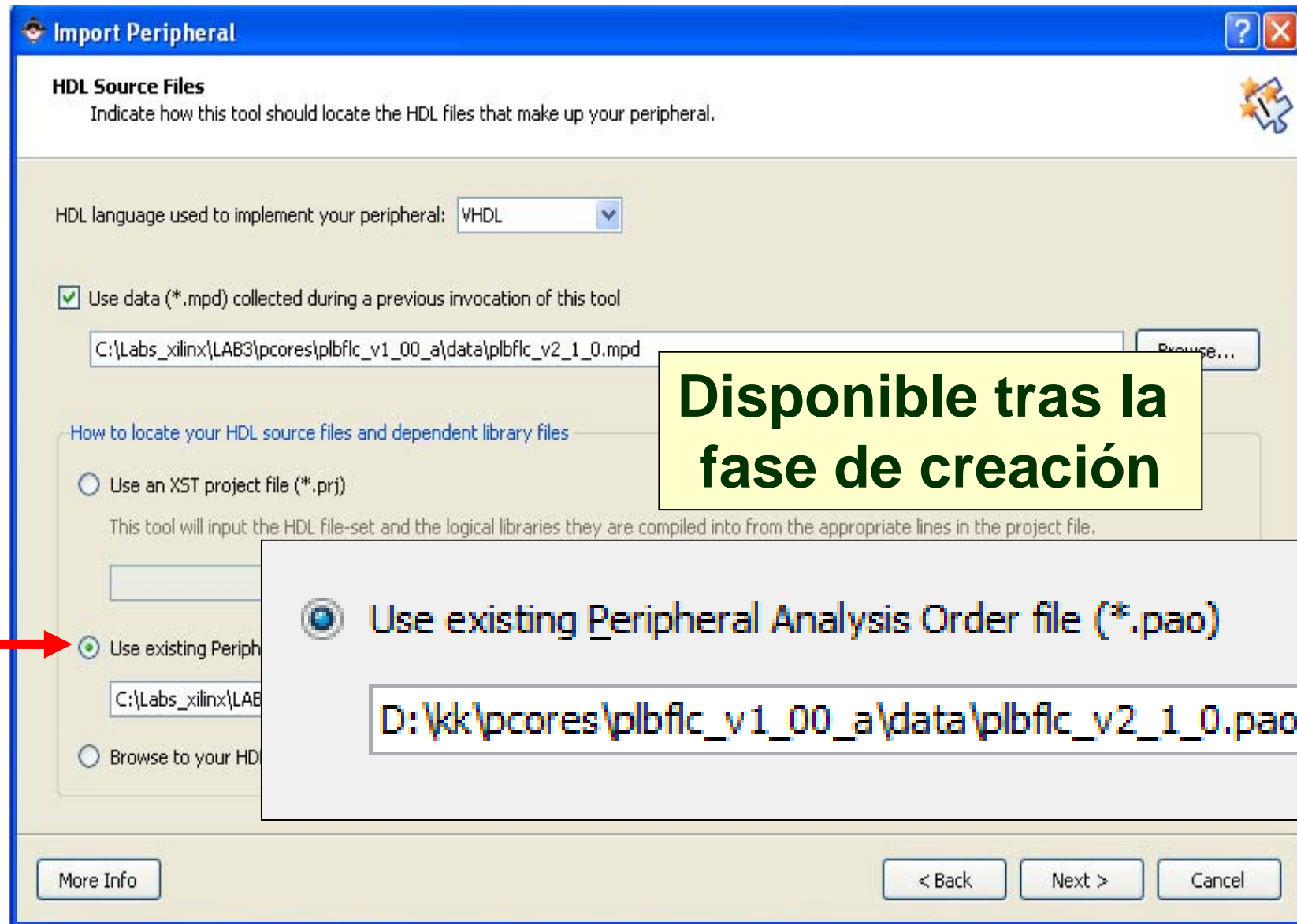
More Info < Back Next > Cancel

Disponible tras la fase de creación

Dos opciones:

- Proyecto **XST**
- Utilizando **PAO**

1.- Utilizando fichero .pao



Import Peripheral

HDL Source Files
Indicate how this tool should locate the HDL files that make up your peripheral.

HDL language used to implement your peripheral: VHDL

☒ Use data (*.mpd) collected during a previous invocation of this tool

C:\Labs_xilinx\LAB3\pcres\plbflc_v1_00_a\data\plbflc_v2_1_0.mpd Browse...

How to locate your HDL source files and dependent library files

☐ Use an XST project file (*.prj)
This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.

☒ Use existing Peripheral Analysis Order file (*.pao)

C:\Labs_xilinx\LAB3\pcres\plbflc_v1_00_a\data\plbflc_v2_1_0.pao

☐ Browse to your HDL source files and dependent library files

More Info < Back Next > Cancel

Disponible tras la fase de creación

Fichero .pao

```
#####  
##  
## Filename:      D:\kk\pcores/plbflc v1 00 a/data/plbflc v2 1 0.pao  
##  
## Description:    Peripheral Analysis Order  
##  
## Date:   Sun Nov 22 09:51:20 2015 (by Create and Import Peripheral Wizard)  
##  
#####
```

```
lib proc_common_v3_00_a proc_common_pkg vhd1  
lib proc_common_v3_00_a ipif_pkg vhd1  
lib proc_common_v3_00_a or_muxcy vhd1  
lib proc_common_v3_00_a or_gate128 vhd1  
lib proc_common_v3_00_a family_support vhd1  
lib proc_common_v3_00_a pselect_f vhd1  
lib proc_common_v3_00_a counter_f vhd1
```

Bibliotecas Xilinx

```
lib plbv46_slave_single_v1_01_a plb_address_decoder vhd1  
lib plbv46_slave_single_v1_01_a plb_slave_attachment vhd1  
lib plbv46_slave_single_v1_01_a plbv46_slave_single vhd1  
lib plbflc_v1_00_a user_logic vhd1  
lib plbflc_v1_00_a plbflc vhd1
```

Plantillas para el módulo IP

2.- Utilizando proyecto XST

Import Peripheral

HDL Source Files
Indicate how this tool should locate the HDL files that make up your peripheral.

HDL language used to implement your peripheral:

☒ Use data (*.mpd) collected during a previous invocation of this tool
D:\kk\pcores\plbfc_v1_00_a\data\plbfc_v2_1_0.mpd

How to locate your HDL source files and dependent library files

☒ Use an XST project file (*.prj)
This tool will input the

D:\kk\pcores\plbfc_v1_00_a\dev\projnav\plbfc.prj

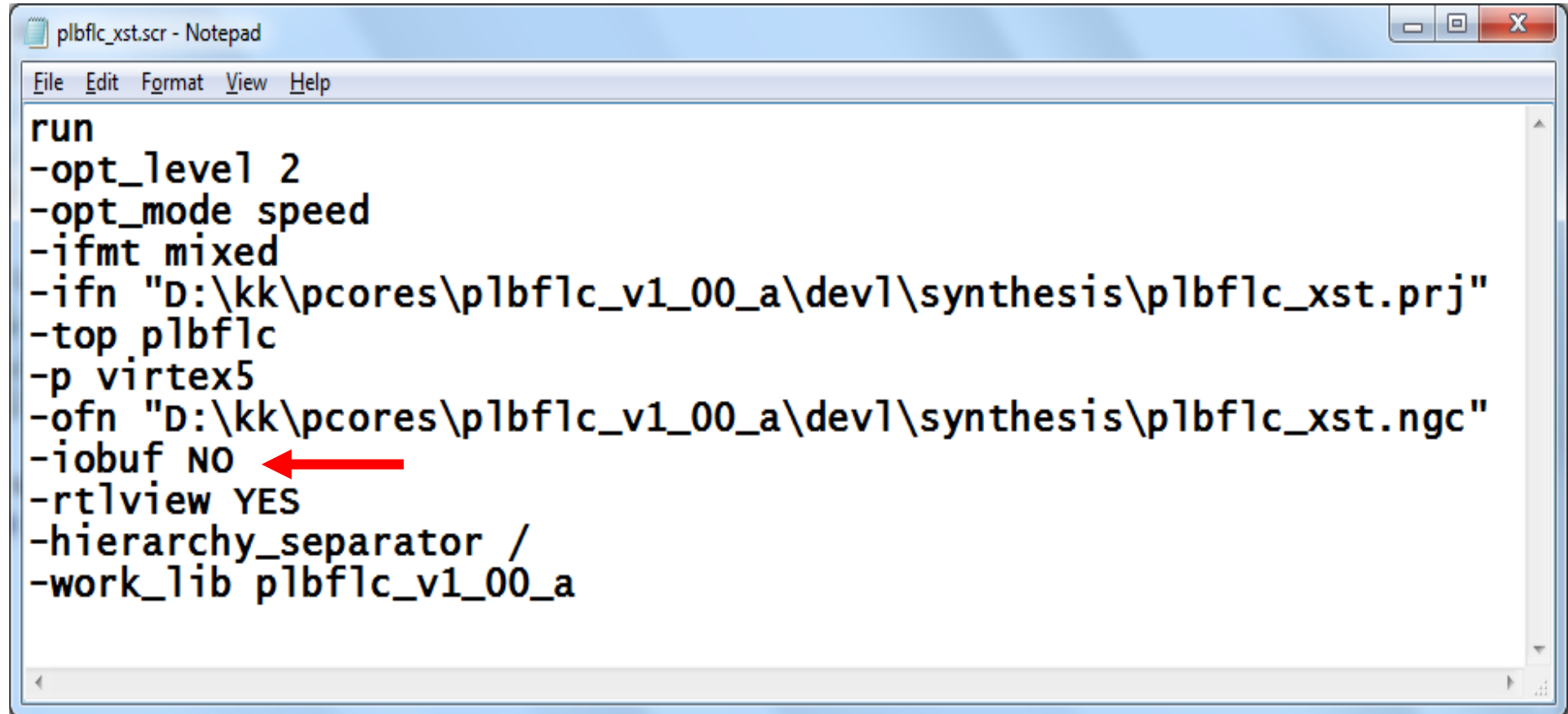
**Disponible tras la
fase de creación**

☒ Use an XST project file (*.prj)

This tool will input the HDL file-set and the logical libraries they

D:\kk\pcores\plbfc_v1_00_a\dev\projnav\plbfc.prj

Opciones de síntesis XST

A screenshot of a Notepad window titled 'plbflc_xst.scr - Notepad'. The window contains a list of XST synthesis options. A red arrow points to the '-iobuf NO' option.

```
run
-opt_level 2
-opt_mode speed
-ifmt mixed
-ifn "D:\kk\pcores\plbflc_v1_00_a\dev1\synthesis\plbflc_xst.prj"
-top plbflc
-p virtex5
-ofn "D:\kk\pcores\plbflc_v1_00_a\dev1\synthesis\plbflc_xst.ngc"
-iobuf NO
-rtlview YES
-hierarchy_separator /
-work_lib plbflc_v1_00_a
```

Inclusión de ficheros de usuario (I)

Import Peripheral

HDL Analysis Information
Indicate the HDL analyze order and the logical libraries your HDL files are compiled into.

Use the buttons on the right to add and remove files, indicate logical libraries and set the HDL analyze order. N

	Language	Logical Library	HDL Source File Path
2	vhdl	proc_common_v3_00_a	C:\Xilinx\12.4\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\proc_common_v
3	vhdl	proc_com	EDK\hw\XilinxProcessorIPLib\pcores\proc_common_v
4	vhdl	proc_com	EDK\hw\XilinxProcessorIPLib\pcores\proc_common_v
5	vhdl	proc_com	EDK\hw\XilinxProcessorIPLib\pcores\proc_common_v
6	vhdl	proc_common_v3_00_a	C:\Xilinx\12.4\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\proc_common_v
7	vhdl	proc_common_v3_00_a	C:\Xilinx\12.4\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\proc_common_v
8	vhdl	plbv46_slave_single_v1_01_a	C:\Xilinx\12.4\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\plbv46_slave_si
9	vhdl	plbv46_slave_single_v1_01_a	C:\Xilinx\12.4\ISE_DS\EDK\hw\XilinxProcessorIPLib\pcores\plbv46_slave_si
10	vhdl	plbflc_v1_00_a	C:\Labs_xilinx\LAB3\pcores\plbflc_v1_00_a\hdl\vhdl\user_logic.vhd
11	vhdl	plbflc_v1_00_a	C:\Labs_xilinx\LAB3\pcores\plbflc_v1_00_a\hdl\vhdl\plbflc.vhd
12	vhdl	plbv46_slav	essorIPLib\pcores\plbv46_slave_si

Elementos de biblioteca

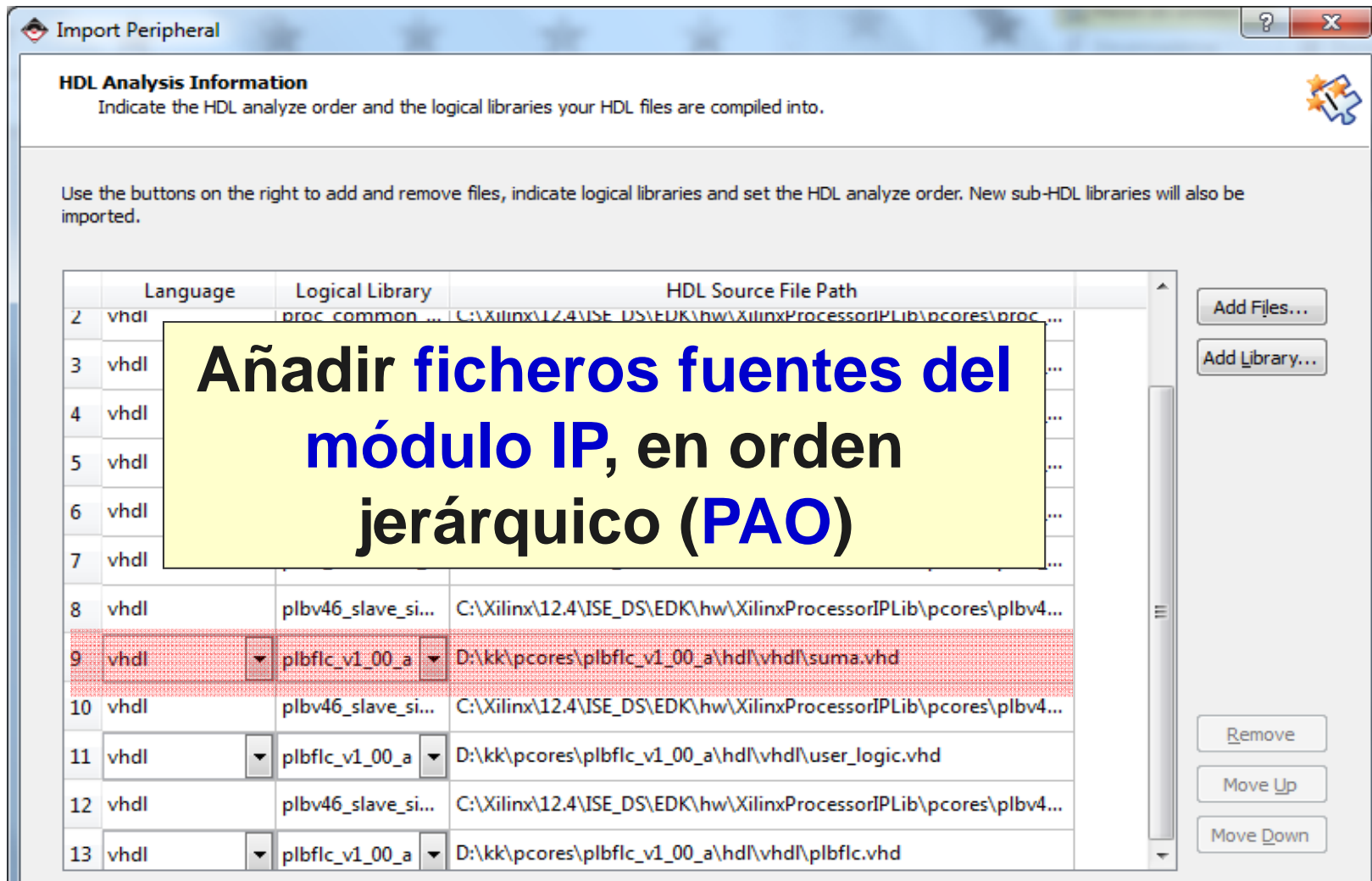
Plantillas VHDL

Añadir fuentes IP

Orden de análisis

Buttons: Add Files..., Add Library..., Remove, Move Up, Move Down, More Info, < Back, Next >, Cancel

Inclusión de ficheros de usuario (II)



Si se desarrolló el **proyecto ISE**, ya aparecen añadidos y en orden jerárquico.

Tipo de bus

Import Peripheral

Bus Interfaces
Identify the bus interfaces supported by your peripheral.

Select the bus interface(s) supported by your peripheral or indicate if there is no applicable bus interface.

A bus interface is a group of related interface ports distinguished by your peripheral or indicate if there is no applicable bus interface.

☒ **Select bus interface(s)**

AXI bus interface

☐ AXI4Lite

☒ Master

☐ Slave

Processor Local Bus (version 4.6) interface

☐ PLBV46 Master (MPLB)

☐ Generate burst

☒ **PLBV46 Slave (SPLB)**

Device Control Register bus interface

☐ DCR Slave (SDCR)

Complex Link bus interface

Master (MFSL)

Slave (SFSL)

< Back Next > Cancel

Nombres asignados en la **fase de creación** y **detectados** de forma automática

Puertos (señales) del bus

SPLB : Port
Define the SPLB bus interface port(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise indicate the ports that correspond to the bus connectors.

Bus Interface Port(s): SPLB

	PLB Bus Connect	Your Port
1	SPLB_Clk	SPLB_Clk
2	SPLB_Rst	SPLB_Rst
3	PLB_abort	PLB_abort
4	PLB_ABus	PLB_ABus
5	PLB_UABus	PLB_UABus
6	PLB_BE	PLB_BE

ATTENTION

The Wizard has successfully extracted bus interface ports for SPLB by applying signal naming convention.

Normalmente no requiere modificación alguna.

Parámetros

Import Peripheral

SPLB : Parameter
Define the SPLB bus interface parameter(s) for this peripheral.

The SPLB bus interface is defined by a predefined set of ports and parameters. If your peripheral follows the standard naming conventions, this tool has automatically done the selections for you. Otherwise check off the values.

☒ **Register Space**

Parameter determine base address: C_BASEADDR

Parameter determine high address: C_HIGHADDR

☐ **Memory Space**

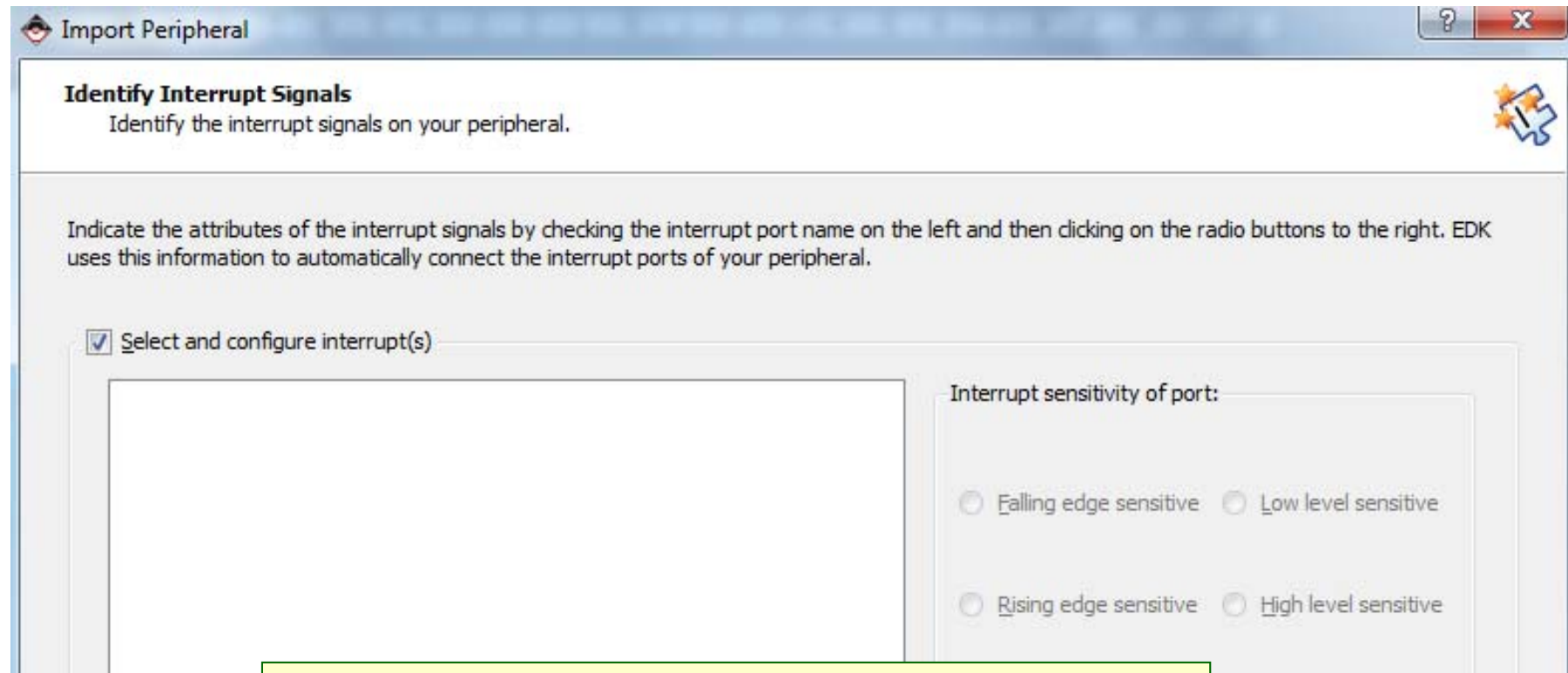
Base Address Parameter	High Address Parameter	Cacheable
------------------------	------------------------	-----------

Add
Remove

Dependen de los **servicios**
seleccionados para el módulo IP



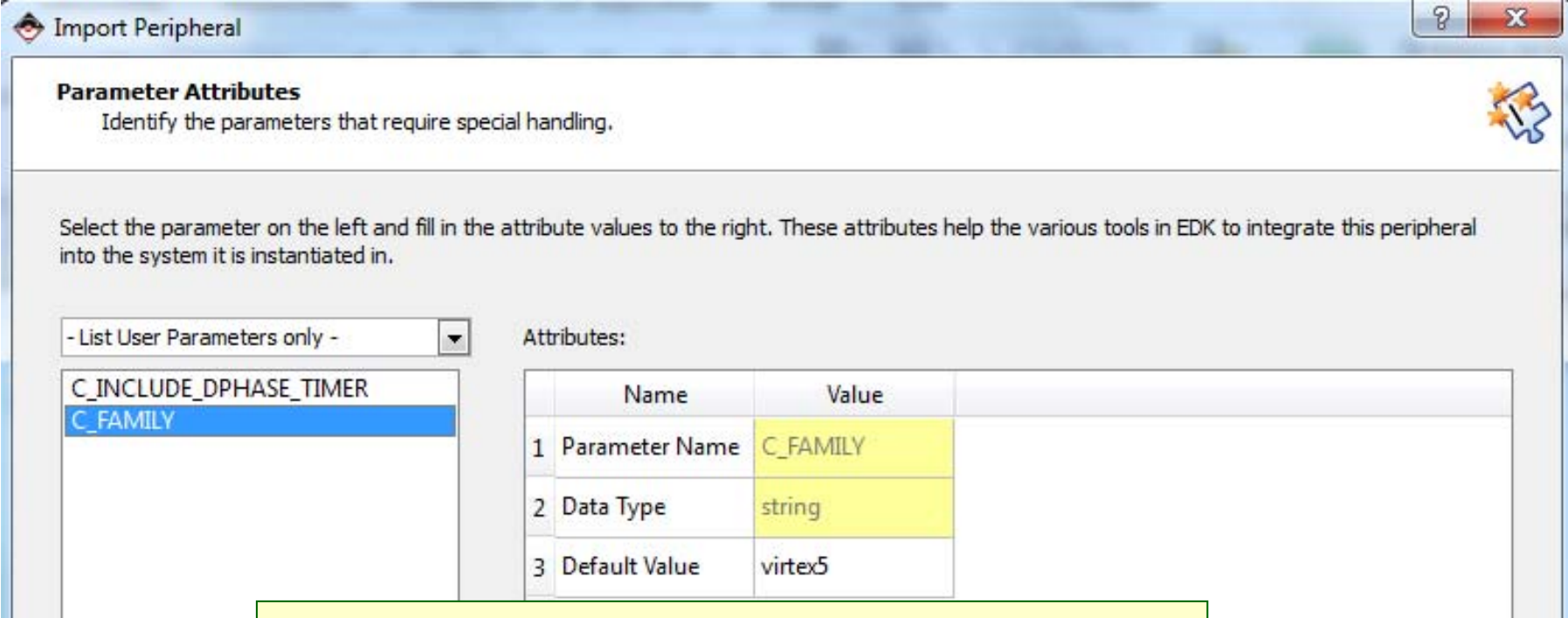
Señales de interrupción



Dependen de los **servicios seleccionados** para el módulo IP



Atributos



Import Peripheral

Parameter Attributes
Identify the parameters that require special handling.

Select the parameter on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to integrate this peripheral into the system it is instantiated in.

- List User Parameters only -

C_INCLUDE_DPHASE_TIMER
C_FAMILY

Attributes:

	Name	Value
1	Parameter Name	C_FAMILY
2	Data Type	string
3	Default Value	virtex5

Dependen de los **servicios seleccionados** para el módulo IP



Puertos

Port Attributes
Identify the ports that require special handling.

Select the port on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to integrate this peripheral into the system it is instantiated in.

- List User Ports only -

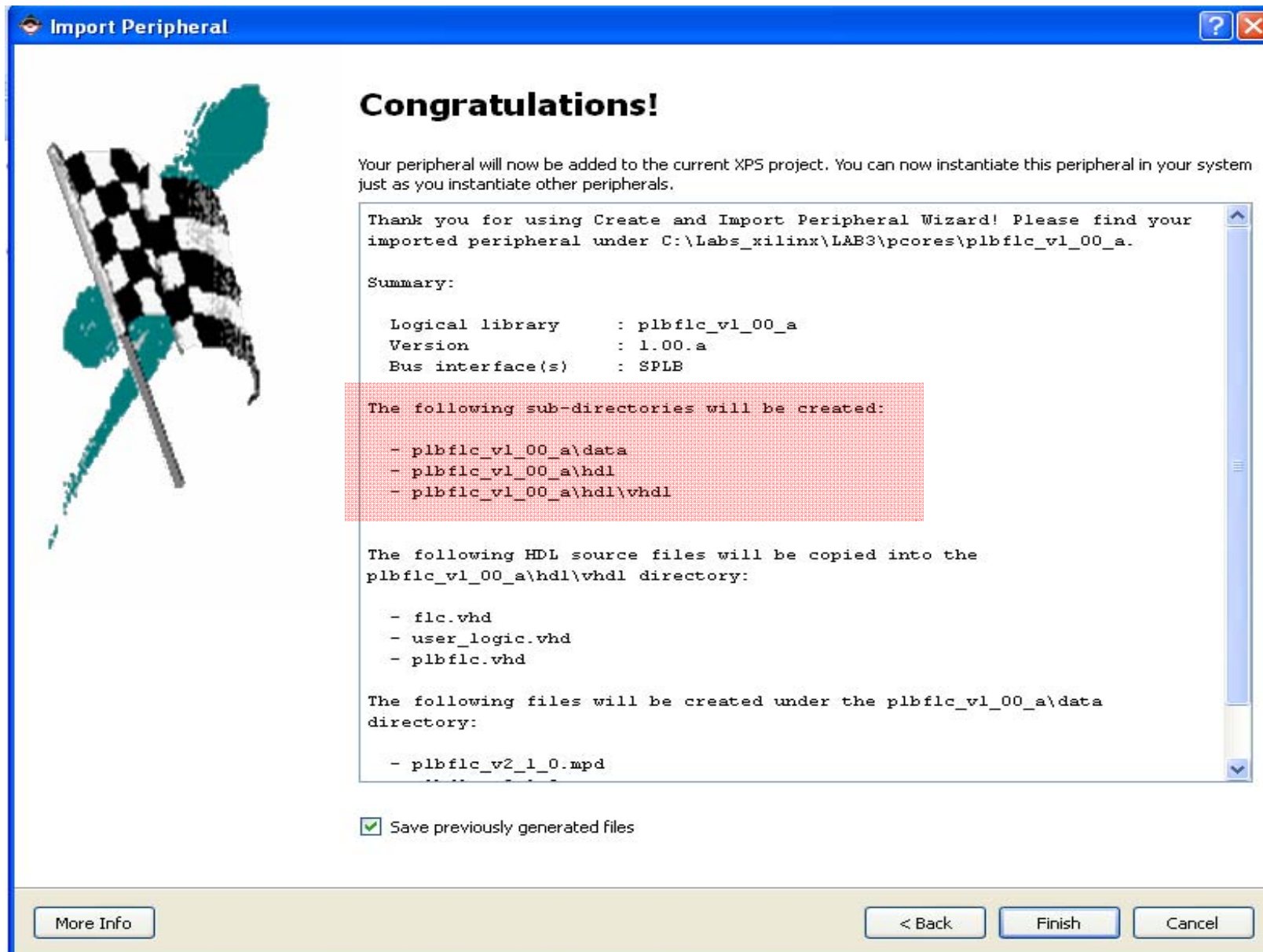
Attributes:

	Name	Value
1	Port Name	
2	Direction Mode	
3	Default Connec...	
4	Vector D...	

Aquí se incluyen (*de existir*)
las **señales externas**
propias del HW añadido

Tienen que haberse añadido como **puertos** en
las **entidades** del *user_logic* y del *top*

Fin del proceso



Empleo de periféricos de usuario

The image shows two windows from the Xilinx design tools. The 'IP Catalog' window on the left displays a tree of IP categories. The 'Project Local PCores' category is expanded, and the 'USER' sub-category is selected, which contains a 'PLBFLC' IP core. The 'XPS Core Config - plbflc_1 - plbflc_v1_00_a' window on the right shows the configuration parameters for this core. The 'Buses' tab is active, showing parameters like C_BASEADDR, C_HIGHADDR, C_INCLUDE_DPHASE_TIMER, C_SPLB_AWIDTH, C_SPLB_CLK_PERIOD_PS, C_SPLB_DWIDTH, and C_SPLB_MID_WIDTH. The 'C_SPLB_CLK_PERIOD_PS' parameter is set to 10,000. A red box highlights the 'USER' category in the IP Catalog, and a blue box highlights the 'PLBFLC' core. A green box highlights the 'Make This IP Local' button at the bottom of the IP Catalog window.

IP Catalog

Description

- EDK Install
- Analog
- Bus and Bridge
- Clock, Reset and Interrupt
- Communication High-Speed
- Communication Low-Speed
- DMA and Timer
- Debug
- General Purpose IO
- IO Modules
- Interprocessor Communication
- Memory and Memory Controller
- PCI
- Peripheral Controller
- Processor
- Utility
- Project Local PCores**
 - USER**
 - PLBFLC**

XPS Core Config - plbflc_1 - plbflc_v1_00_a

All Buses HDL PDF

C_BASEADDR 0xFFFFFFFF

C_HIGHADDR 0x00000000

C_INCLUDE_DPHASE_TIMER ☐

C_SPLB_AWIDTH 32

☒ C_SPLB_CLK_PERIOD_PS 10,000

C_SPLB_DWIDTH 128

C_SPLB_MID_WIDTH 3

OK Cancel Help

Add IP

View MP

Make This IP Local

Empleo de periféricos de usuario

The screenshot displays the Xilinx Platform Studio interface in the 'System Assembly View'. The 'IP Catalog' on the left lists various components, with 'Project Local PCores' expanded to show 'USER' and 'PLBFLC'. A red circle highlights the 'PLBFLC' entry, with a red arrow pointing to the 'plbflc_0' component in the 'Bus Interfaces' table on the right. The table lists various IP blocks, including 'plbflc_0' which is highlighted with a red box. Below the table, a red box highlights the 'plbflc_0' component in the system assembly, showing its connection to the 'SPLB' bus and the 'mb_plb' port.

Name	Bus Name	IP Type	IP Version
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.05.a
microblaze_0		microblaze	8.00.b
lmb_bram		bram_block	1.00.a
dlmb_cntlr		lmb_bram_i...	2.10.b
ilmb_cntlr		lmb_bram_i...	2.10.b
DDR2_SDRAM		mpmc	6.02.a
mdm_0		mdm	2.00.a
plbflc_0	SPLB	plbflc	1.00.a
BTNs_4Bit		xps_gpio	2.00.a
DIPs_4Bit		xps_gpio	2.00.a
LEDs_8Bit		xps_gpio	2.00.a
RS232_DCE		xps_uartlite	1.01.a
clock_gener...		clock_gene...	4.01.a
proc_sys_re...		proc_sys_re...	3.00.a



Empleo de periféricos de usuario

Bus Interfaces								Ports	Addresses	Generate Addresses	
Instance	Base Name	Base Address	High Address	Size	Bus Interfa	Bus Name	Lock				
microblaze_0's Ad...											
dlmb_cntlr	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	dlmb	<input checked="" type="checkbox"/>				
ilmb_cntlr	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	ilmb	<input checked="" type="checkbox"/>				
DDR2_SDRAM	C_MPMC_BAS...	0x44000000	0x47FFFFFF	64M	XCL0:XC...	microblaz...	<input checked="" type="checkbox"/>				
LEDs_8Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>				
DIPs_4Bit	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>				
BTNs_4Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>				
RS232_DCE	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>				
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>				
plbflc_0	C_BASEADDR	0xC6400000	0xC640FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>				



Inclusión en la plataforma HW/SW

system.mhs

MHS

```
204 BEGIN xps_intc
205   PARAMETER INSTANCE = xps_intc_0
206   PARAMETER HW_VER = 2.01.a
207   PARAMETER C_BASEADDR = 0x81800000
208   PARAMETER C_HIGHADDR = 0x8180ffff
209   BUS_INTERFACE SPLB = mb_plb
210   PORT Irq = microblaze_0_INTERRUPT
211   PORT Intr = xps_timer_0_interrupt
212 END
213
214 BEGIN plbflc
215   PARAMETER INSTANCE = plbflc_0
216   PARAMETER HW_VER = 1.00.a
217   PARAMETER C_BASEADDR = 0xc6400000
218   PARAMETER C_HIGHADDR = 0xc640ffff
219   BUS_INTERFACE SPLB = mb_plb
220 END
221
222
```

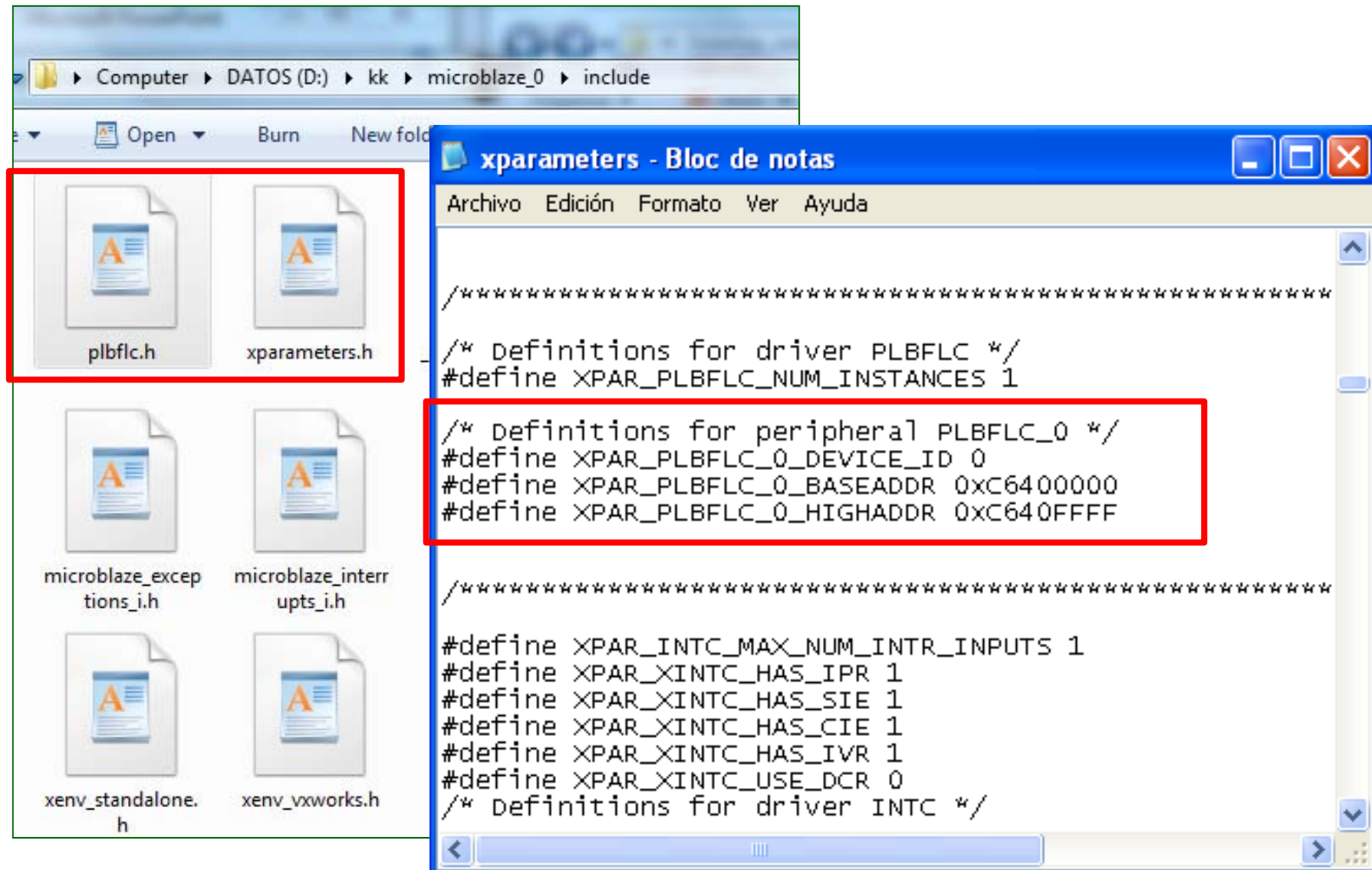
system.mss

MSS

```
87 END
88
89 BEGIN DRIVER
90   PARAMETER DRIVER_NAME = tmrctr
91   PARAMETER DRIVER_VER = 2.02.a
92   PARAMETER HW_INSTANCE = xps_timer_0
93 END
94
95 BEGIN DRIVER
96   PARAMETER DRIVER_NAME = intc
97   PARAMETER DRIVER_VER = 2.02.a
98   PARAMETER HW_INSTANCE = xps_intc_0
99 END
100
101 BEGIN DRIVER
102   PARAMETER DRIVER_NAME = plbflc
103   PARAMETER DRIVER_VER = 1.00.a
104   PARAMETER HW_INSTANCE = plbflc_0
105 END
106
107
```



Parámetros y drivers generados por *LibGen*



Driver *plbflc.h*

```
/****** Include Files *****/

#include "xbasic_types.h"
#include "xstatus.h"
#include "xil_io.h"

/****** Constant Definitions *****/

#define PLBFLC_USER_SLV_SPACE_OFFSET (0x00000000)
#define PLBFLC_SLV_REG0_OFFSET (PLBFLC_USER_SLV_SPACE_OFFSET + 0x00000000)
#define PLBFLC_SLV_REG1_OFFSET (PLBFLC_USER_SLV_SPACE_OFFSET + 0x00000004)
#define PLBFLC_SLV_REG2_OFFSET (PLBFLC_USER_SLV_SPACE_OFFSET + 0x00000008)

/****** Type Definitions *****/

/** Write a value to a PLBFLC register. A 32 bit write is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is written. */

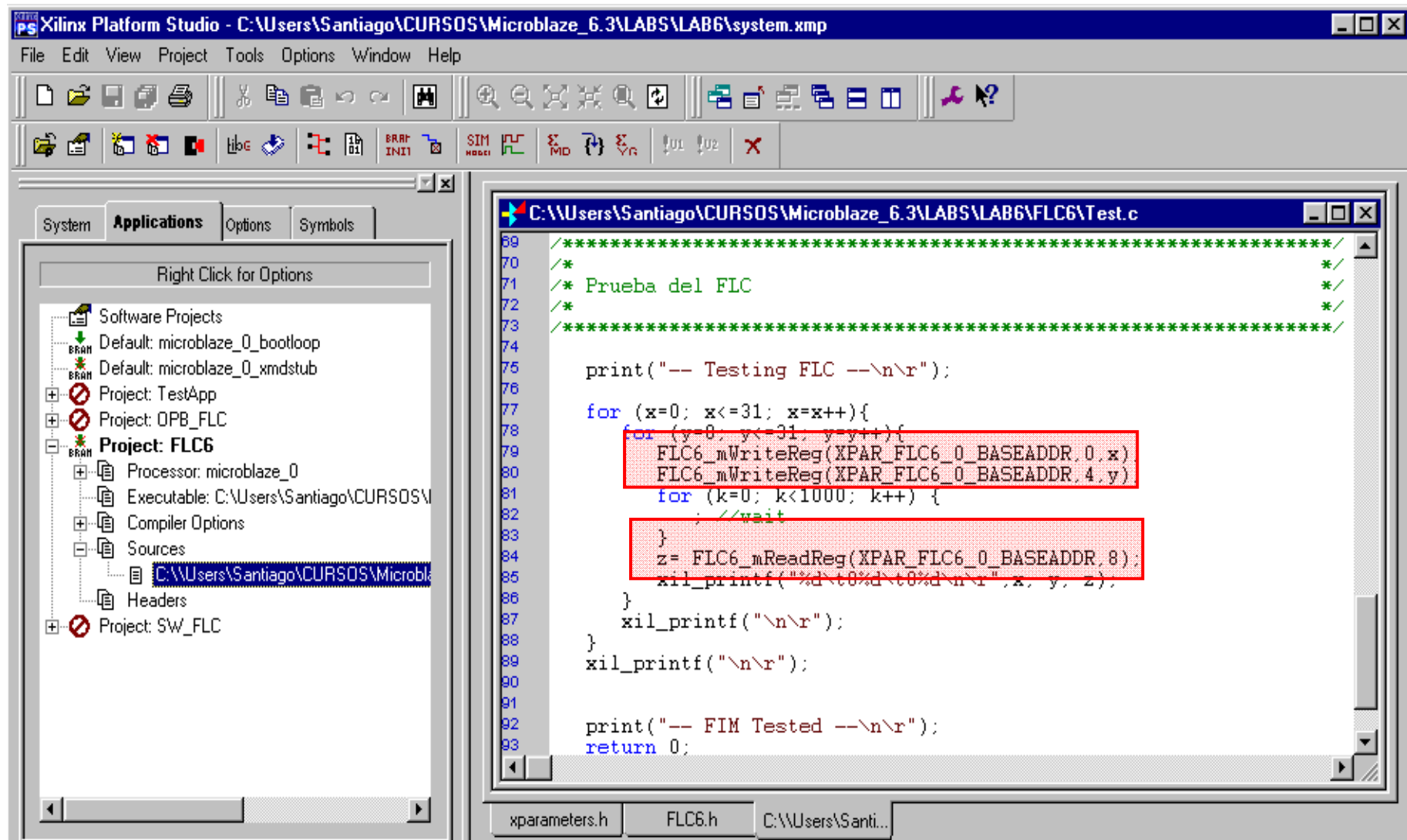
#define PLBFLC_mWriteReg(BaseAddress, RegOffset, Data) \
    Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))

/* Read a value from a PLBFLC register. A 32 bit read is performed.
 * If the component is implemented in a smaller width, only the least
 * significant data is read from the register. The read value
 * will be read as 0.
 * @return Data is the data from the register. */

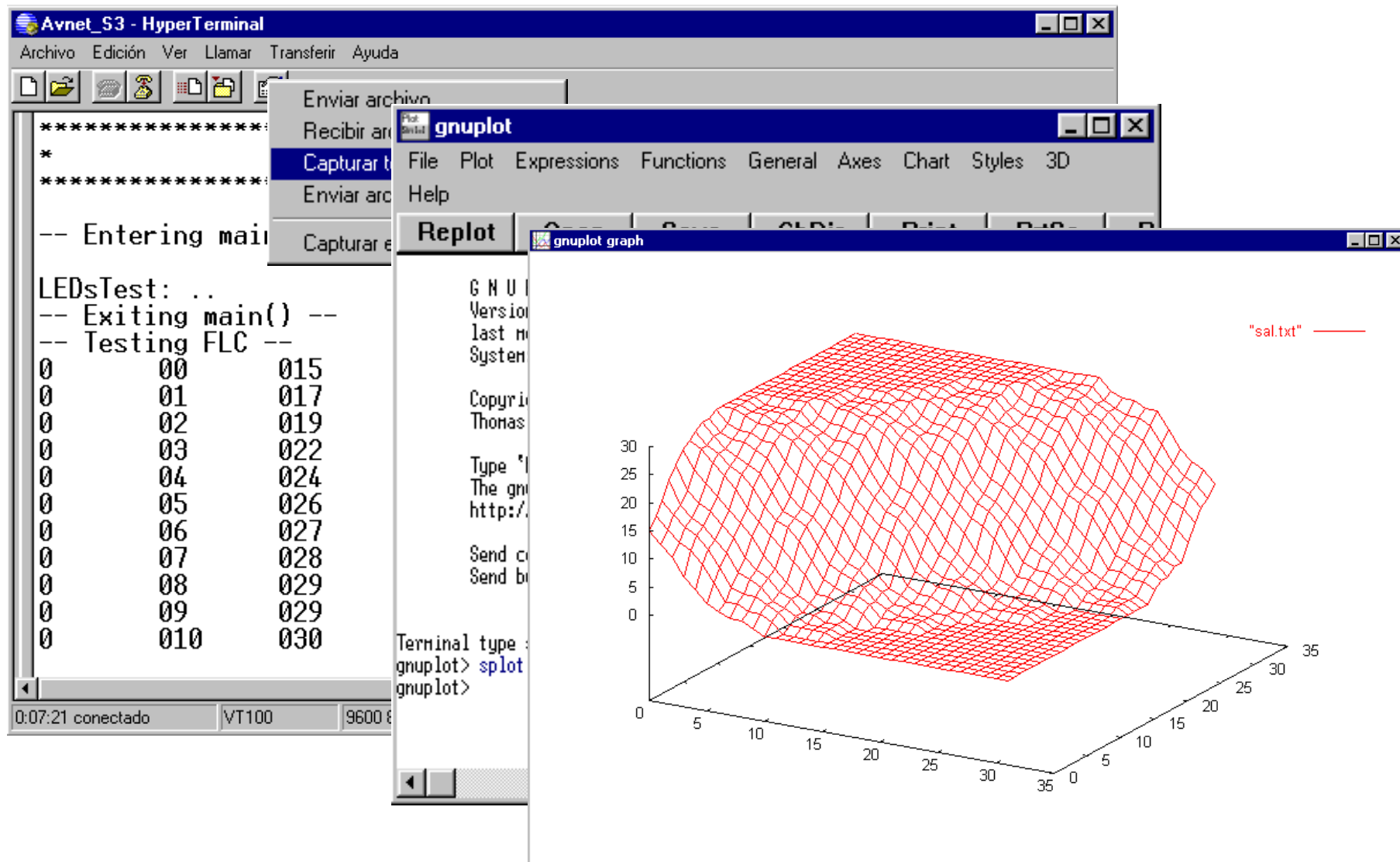
#define PLBFLC_mReadReg(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (RegOffset))
```

Las funciones dependen de los servicios seleccionados

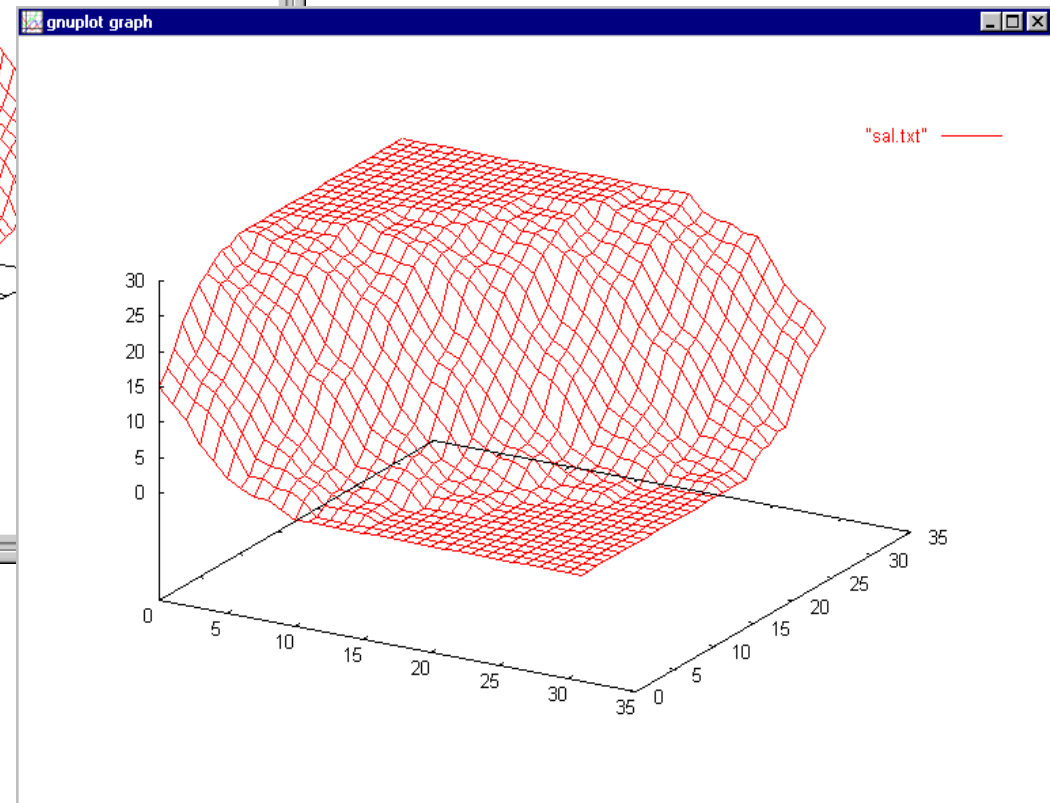
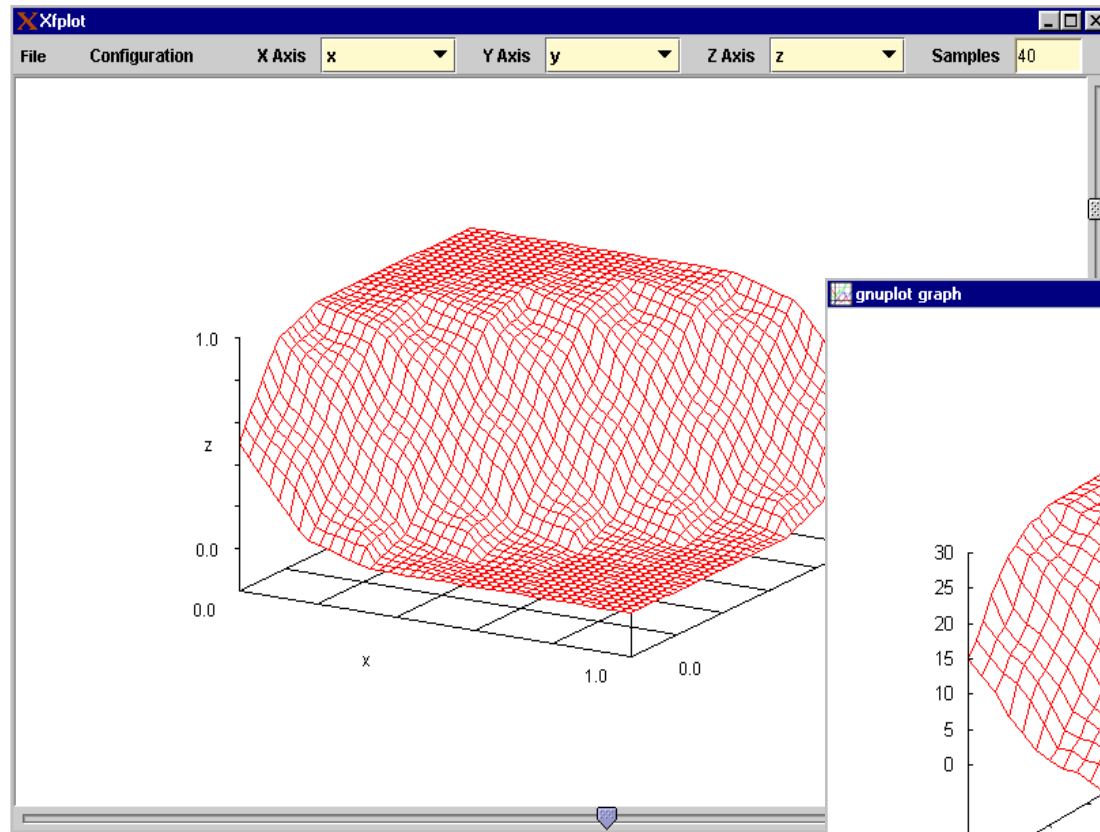
Desarrollo de software de aplicación



Ejecución del programa



Comparación de resultados



Documentación

○ Manuales

- *Processor IP Reference Guide*
- *Driver Reference Guide*
- *Embedded System Tools Ref. Manual*
→ *Create/Import Peripheral Wizard*
- *Embedded System Tools Ref. Manual*
→ *Platform Specification Utility*

○ Soporte Web

- EDK
 - <http://www.support.xilinx.com/edk>



Inclusión del módulo IP en *XPS*

- ❑ Existen dos modos para integrar un **módulo IP** en *Xilinx Platform Studio*:
 - Como una “**lista de conexionado**” (*netlist*)
 - Sintetizado con *XST* con el resto del sistema de procesado
 - Requiere ficheros:
 - **MPD**: *Microprocessor Peripheral Description*
 - **PAO**: *Peripheral Analyze Order*
 - Como una “**caja negra**” (*blackbox*)
 - Previamente *sintetizado* con cualquier herramienta de síntesis
 - Requiere ficheros:
 - **MPD**: *Microprocessor Peripheral Description*
 - **BBD**: *Black Box Definition*