



Práctica 5. Desarrollo de una aplicación SW avanzada

Carlos González

Hortensia Mecha

*Dpto. Arquitectura de Computadores y Automática
Univ. Complutense de Madrid*

Objetivos

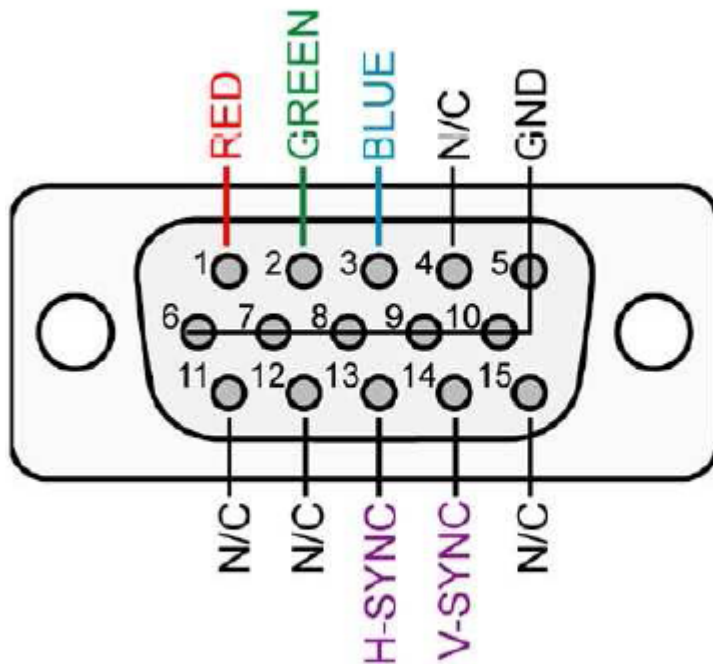
- Comprender el flujo de Diseño de las herramientas de Desarrollo de Sistemas Empotrados (Embedded Development Kit -**EDK**) de Xilinx para la creación e importación de periféricos
- Adquirir soltura en el desarrollo de una aplicación software empotrada que se comuniquen con distintos periféricos



- ■ **Fundamentos VGA**
 - Parte a
 - Añadir periférico VGA
 - Parte b
 - Control de pantalla

Protocolo VGA

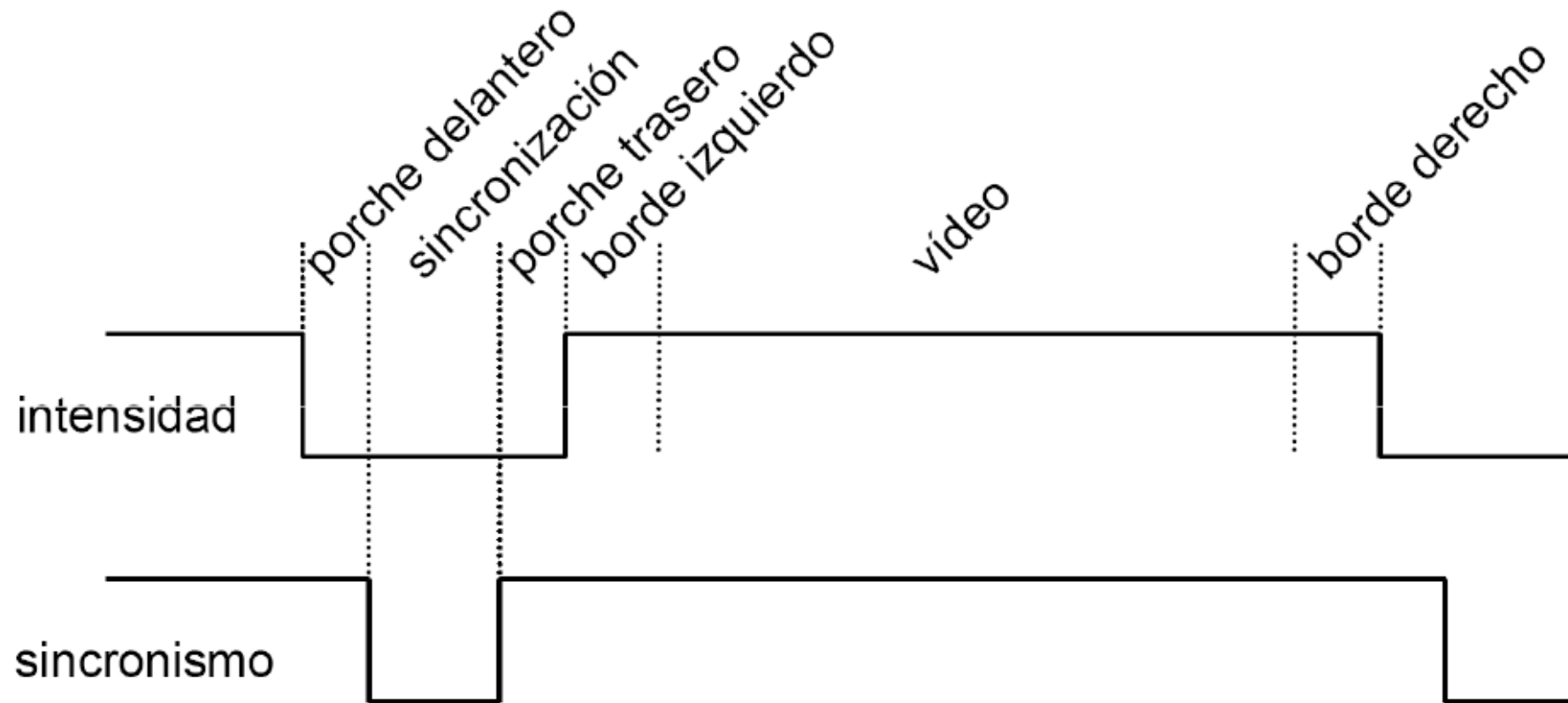
- El protocolo VGA utiliza dos líneas de sincronismo y tres para color



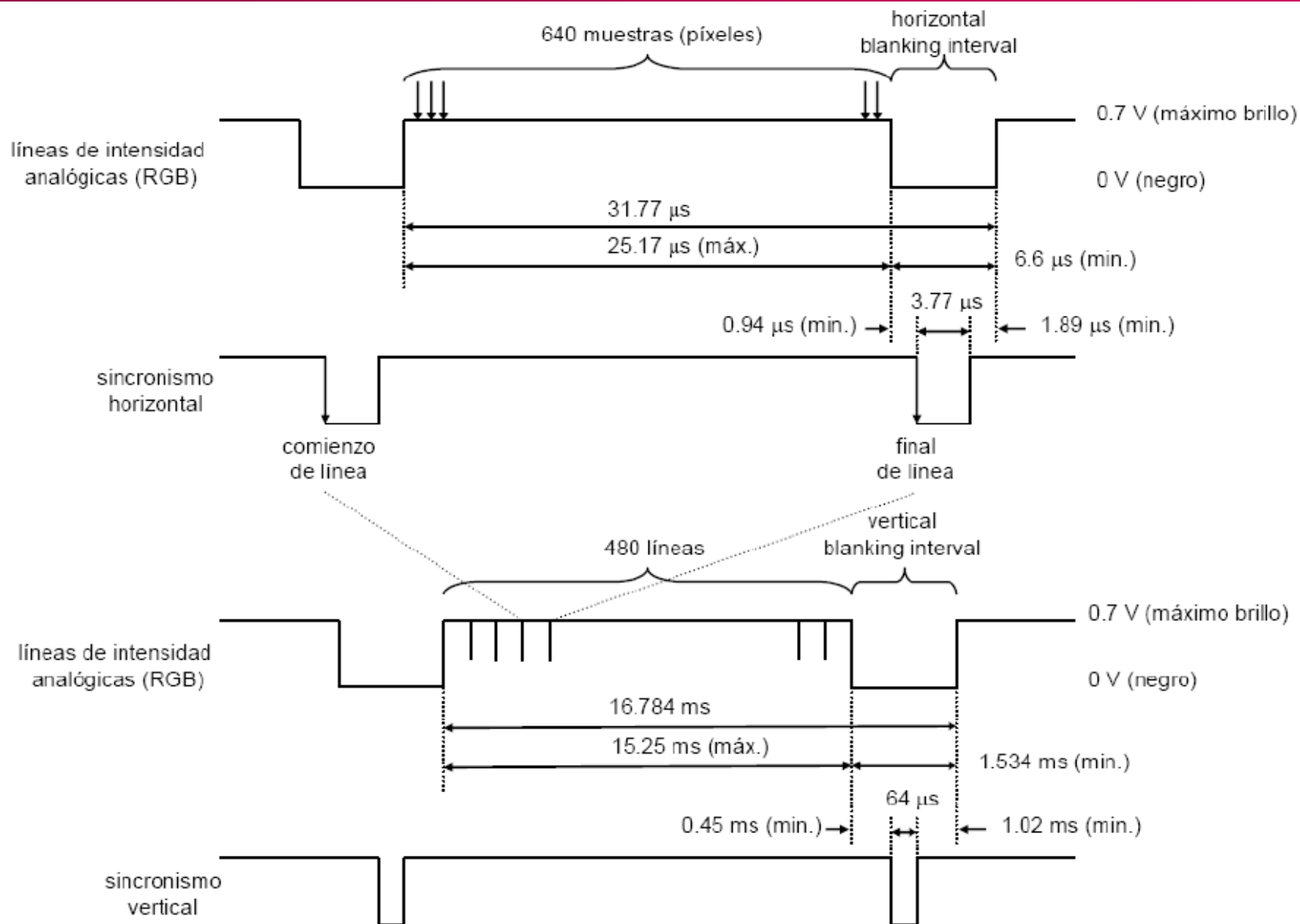
Puerto VGA visto desde el cable

Pin	Nombre	Descripción
1	RED	Canal rojo
2	GREEN	Canal verde
3	BLUE	Canal azul
4	N/C	Sin contacto
5	GND	Tierra
6	GND	Tierra
7	GND	Tierra
8	GND	Tierra
9	GND	Tierra
10	GND	Tierra
11	N/C	Sin contacto
12	N/C	Sin contacto
13	H-SYNC	Sincronización horizontal
14	V-SYNC	Sincronización vertical
15	N/C	Sin contacto

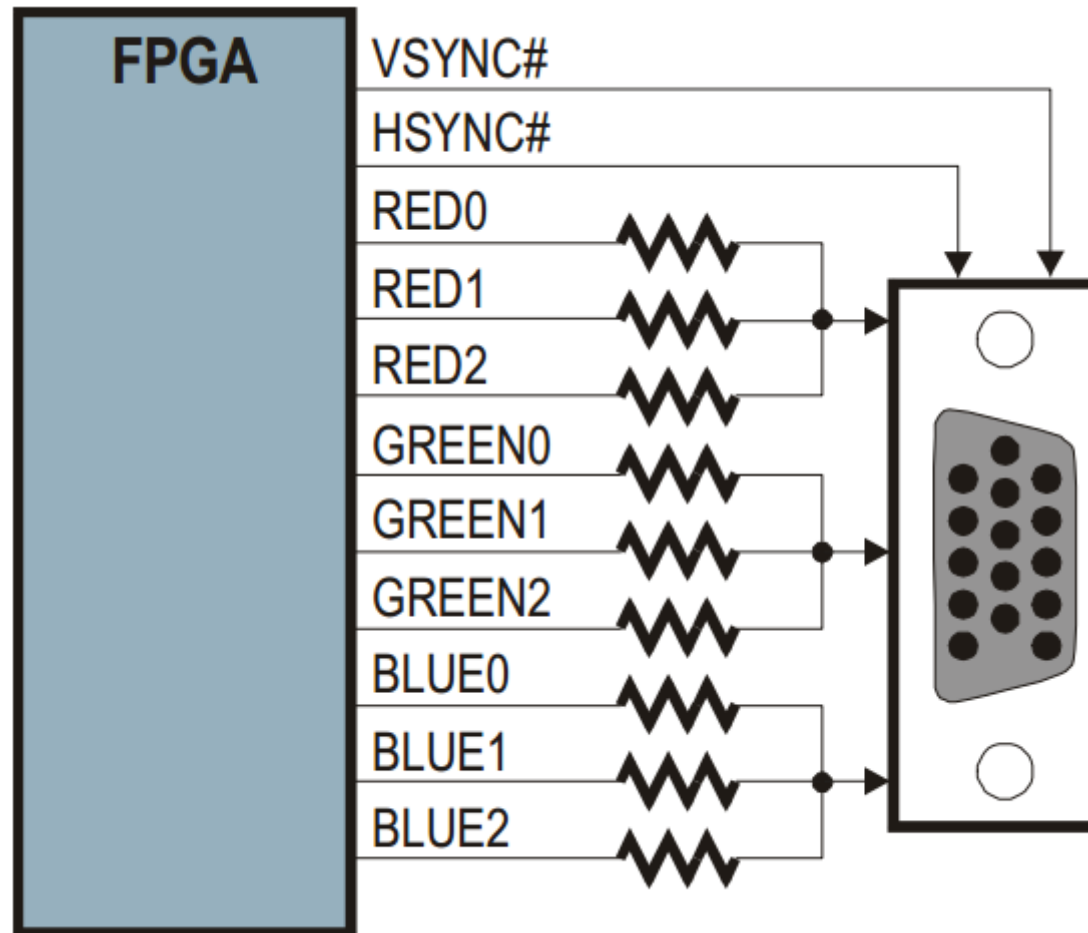
Temporización VGA



Temporización VGA



VGA en las placas de prototipado

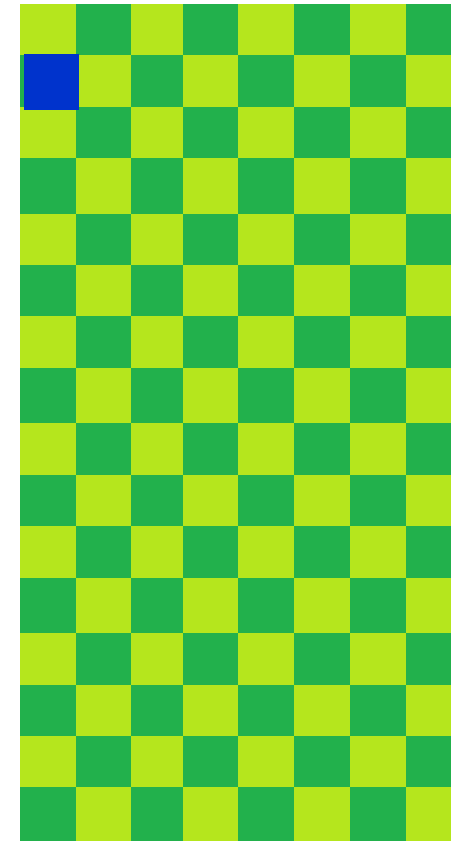
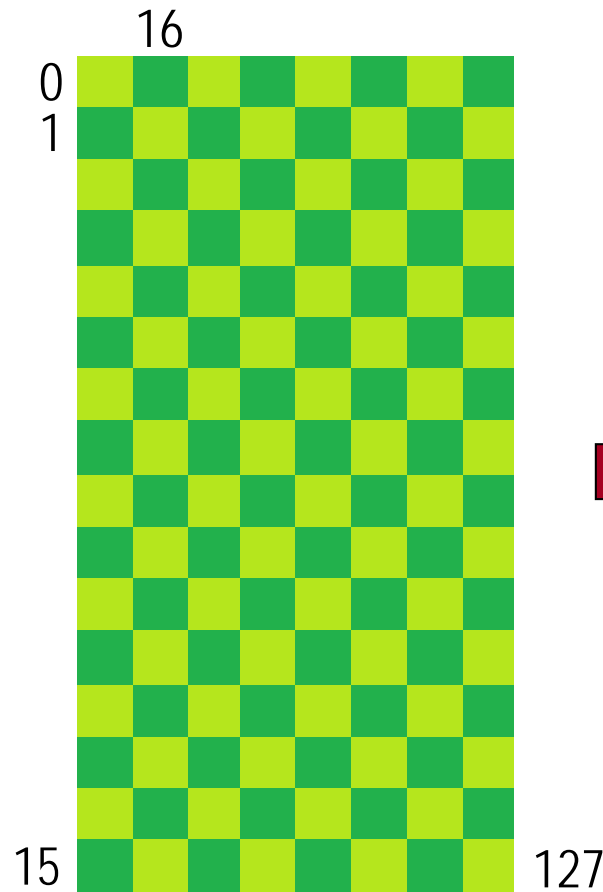


- Fundamentos VGA
- ■ **Parte a**
 - **Añadir periférico VGA**
- Parte b
 - Conecta 4

Añadir periférico VGA

- En este apartado añadiremos un periférico VGA para pintar una matriz de cuadrados de 16 filas y 8 columnas
- Funcionamiento

Rectángulo: 1
Color: Azul



Añadir periférico VGA

- La comunicación del procesador con el periférico se realiza a través de una FIFO mediante el envío de comandos
- Los comandos tienen el siguiente formato
 - 32 bits
 - ✓ 7 LSB → Codifican el rectángulo a colorear
 - ✓ 9 MSB → Codifican el color para el rectángulo
 - ✓ El resto de bits no se utilizan



Partimos del proyecto "base"

Project Information Area

Project: Ap...
Platform:
Project F...
MHS File: lab1b.mhs
MSS File: lab1b.mss
UCF File: data/lab1b.ucf
iMPACT Command File: etc/download.cmd
Implementation Options File: etc/fast_runtime.opt
Bitgen Options File: etc/bitgen.opt
Project D...
Device:
Netlist:
Implementation: XPS (Xflow)
HDL: VHDL
Sim Model: BEHAVIORAL
Reference Files
Log Files
Synth...

System Assembly View

Bus Interfaces | Ports | Addresses

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.a
dmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
lmb_plb		plb_v46	1.02.a
dmb_cntlr		lmb_bram_if_cntlr	2.10.a
ilmb_cntlr		lmb_bram_if_cntlr	2.10.a
lmb_bram		bram_block	1.00.a
debug_module		mdm	1.00.b
xps_gpio_LEDS		xps_gpio	1.00.a
xps_gpio_SWITCHES		xps_gpio	1.00.a
RS232		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.00.a
proc_sys_reset_0		proc_sys_reset	2.00.a

Console Window

Output | Warning | Error

Listo



Creando el periférico con bus PLB

The screenshot shows the Xilinx Platform Studio interface. The 'Hardware' menu is open, and the 'Create or Import Peripheral...' option is highlighted with a red arrow. A yellow box with the text 'Abrir el asistente para crear periféricos' is overlaid on the interface. The 'System Assembly View' is active, showing a list of components and their connections.

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.a
dlimb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb			02.a
lmb_d			10.a
ilmb_d			10.a
lmb_br			00.a
debug			00.b
xps_gpio			00.a
xps_gpio_SWITCHES		xps_gpio	1.00.a
RS232		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.00.a
proc_sys_reset_0		proc_sys_reset	2.00.a



Creando el periférico con bus PLB

Create and Import Peripheral Wizard - Peripheral Flow

Peripheral Flow
Indicate if you want to create a new peripheral or import an existing peripheral.

This tool will help you create templates for a new EDK compliant peripheral, or help you import an existing peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

Select flow

- ☒ Create templates for a new peripheral
- ☐ Import existing peripheral

Flow description
This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

Options

☐ Load an existing .cip settings file (saved from a previous session)

Diagram:

```
graph TD; A[Create Templates] --> B[Implement/Verify]; B --> C[Import to XPS]; B --> A;
```

Vamos a crear el periférico a partir de una plantilla



Creando el periférico con bus PLB

Create Peripheral - Repository or Project

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: Browse...

☒ To an XPS project

Project: Browse...

Peripheral will be placed under:

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

**Creamos el periférico en el proyecto actual.
De forma alternativa, se puede crear en el
repositorio de EDK si el periférico se va a
utilizar en otros proyectos**



Creando el periférico con bus PLB

Create Peripheral - Name and Version

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 as the top HDL design entity.

Name:

Version: 1.00.a

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

Description:

Logical library name: `pantalla_v1_00_a`

All HDL files (either created by you or generated by this tool) that are used to implement this peripheral must be compiled into the logical library name above. Any other referred logical libraries 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.

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



Creando el periférico con bus PLB

Create Peripheral - Bus Interface

Bus Interface
Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

☒ Processor Local Bus (PLB v4.6) ← **Seleccionamos el bus PLB**

☐ Fast Simplex Link (FSL)

ATTENTION
Refer to the following documents to get a better understanding of how user peripherals connect to the CoreConnect(TM) buses (including PLB v4.6 interconnect and OPB/PLB v3.4 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](#)

Note
Xilinx recommends using the new PLB v4.6 bus standard, however, the wizard still supports the OPB and PLB v3.4 bus interfaces.

☐ Enable OPB and PLB v3.4 bus interfaces

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



Creando el periférico con bus PLB

Create Peripheral - IPIF (IP Interface) Services

IPIF (IP Interface) Services
Indicate the IPIF services required by your peripheral.

Your peripheral 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.

Vamos a utilizar una FIFO para enviar órdenes al periférico

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
- ☒ Read/Write FIFO
- ☐ Interrupt control
- ☐ User logic software register
- ☐ User logic memory space

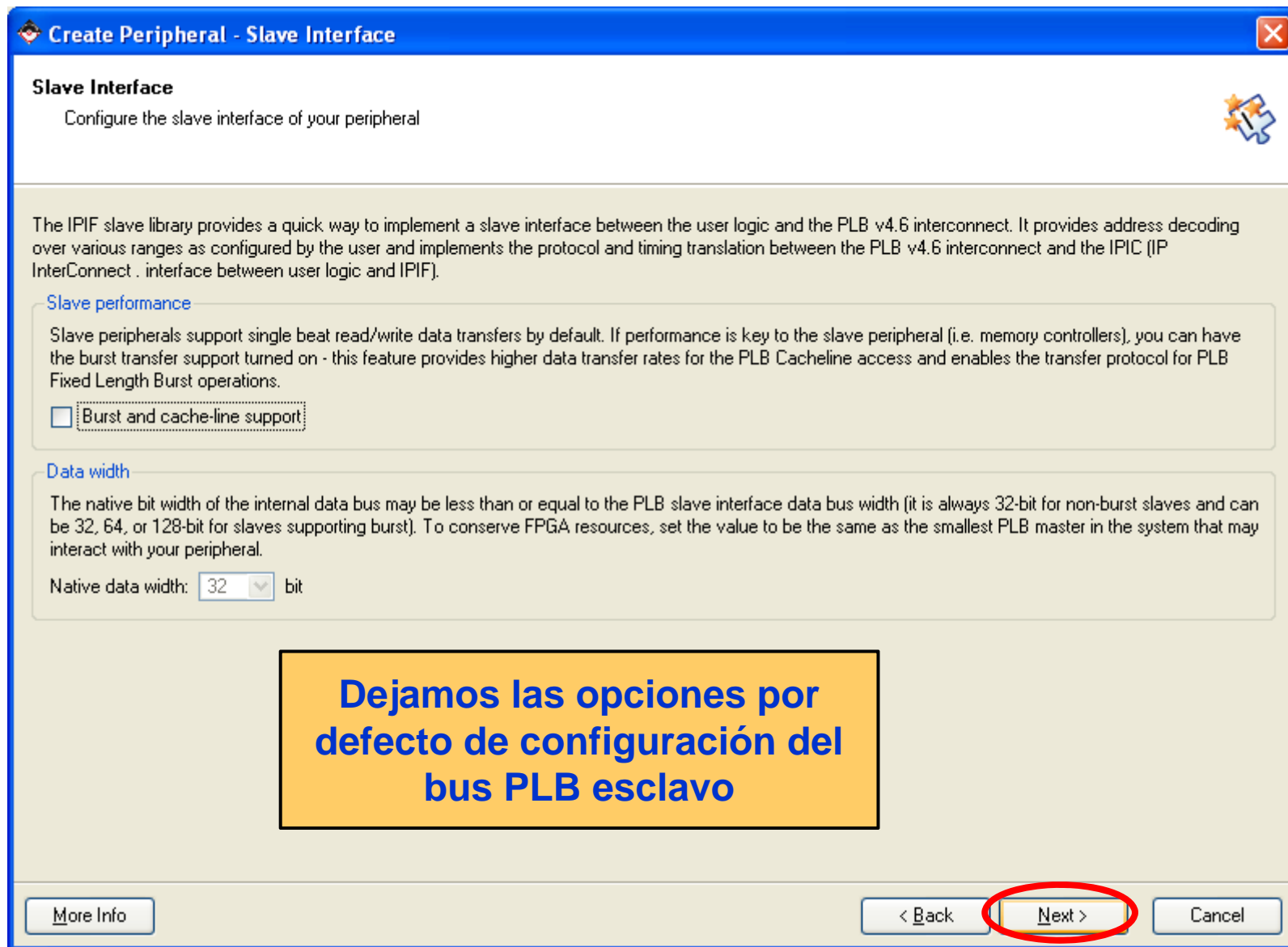
Master service and configuration
Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected).

- ☐ User logic master

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



Creando el periférico con bus PLB



Create Peripheral - Slave Interface

Slave Interface
Configure the slave interface of your peripheral

The IPIF slave library provides a quick way to implement a slave interface between the user logic and the PLB v4.6 interconnect. It provides address 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

Dejamos las opciones por defecto de configuración del bus PLB esclavo

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



Creando el periférico con bus PLB

Create Peripheral - FIFO Service

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

☐ Use packet mode

☐ Use vacancy calculation

Number of Read FIFO entries: 512

☒ Include Write FIFO

☐ Use packet mode

☐ Use vacancy calculation

Number of Write FIFO entries: 512

Write FIFO Block Diagram:

Datasheet

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



Creando el periférico con bus PLB

Create Peripheral - IP Interconnect (IPIC)

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

The diagram shows a yellow box labeled 'Peripheral' containing three sub-blocks: 'PLB v4.6 Slave' (green), 'Other Blocks' (orange), and 'PLB v46 Master' (yellow). Below these is a blue box labeled 'User Logic'. Three vertical double-headed arrows connect the sub-blocks to the User Logic, labeled 'IPIC for slave', 'IPIC for others', and 'IPIC for master' respectively.

Port description

- ☒ 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
- ☒ IP2wFIFO_RdReq
- ☐ IP2wFIFO_RdMark
- ☐ IP2wFIFO_RdRelease
- ☐ IP2wFIFO_RdRestore
- ☒ wFIFO2IP_Data
- ☒ wFIFO2IP_RdAck

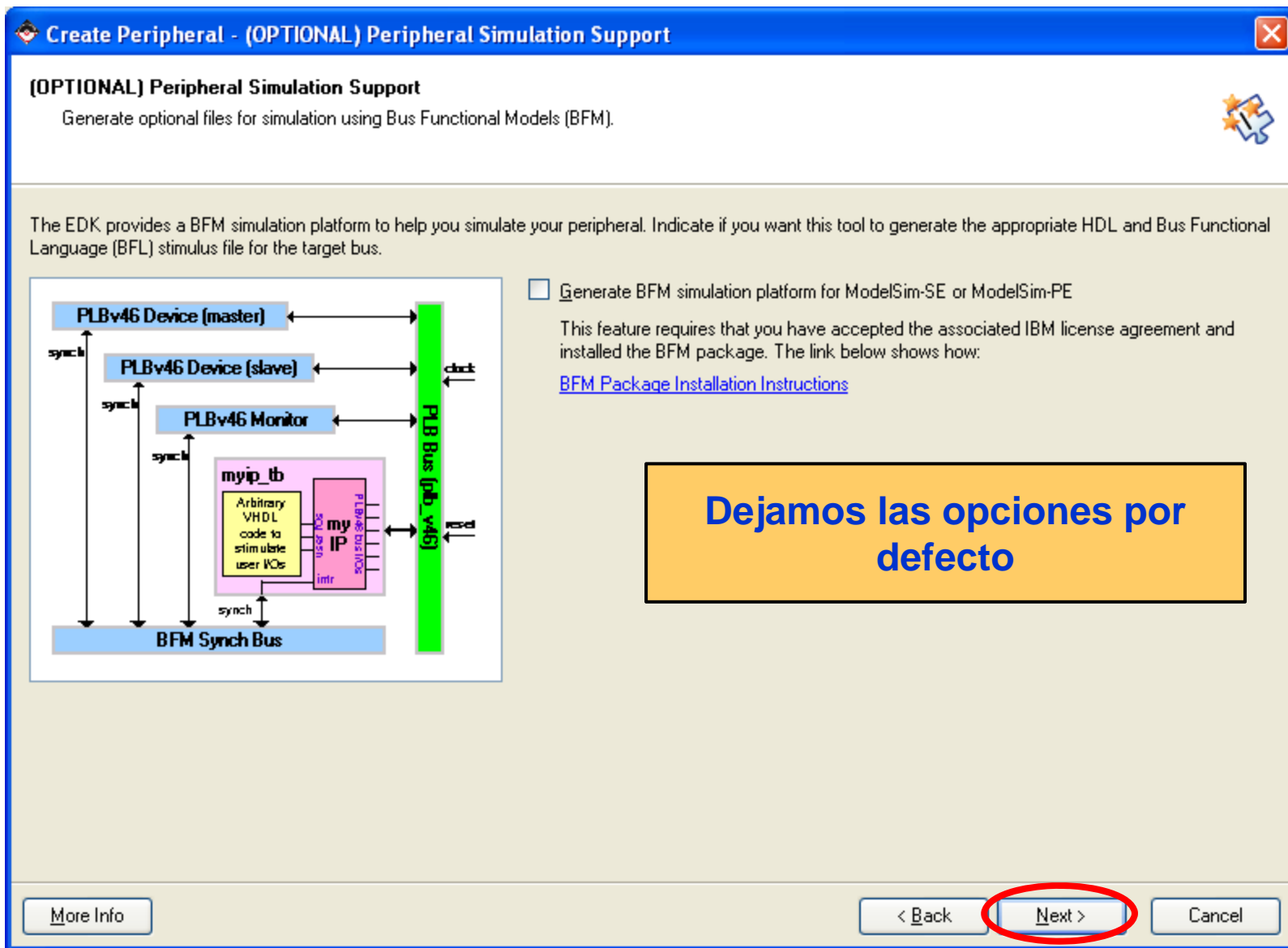
Restore Defaults

Dejamos las opciones por defecto de configuración para las señales del bus

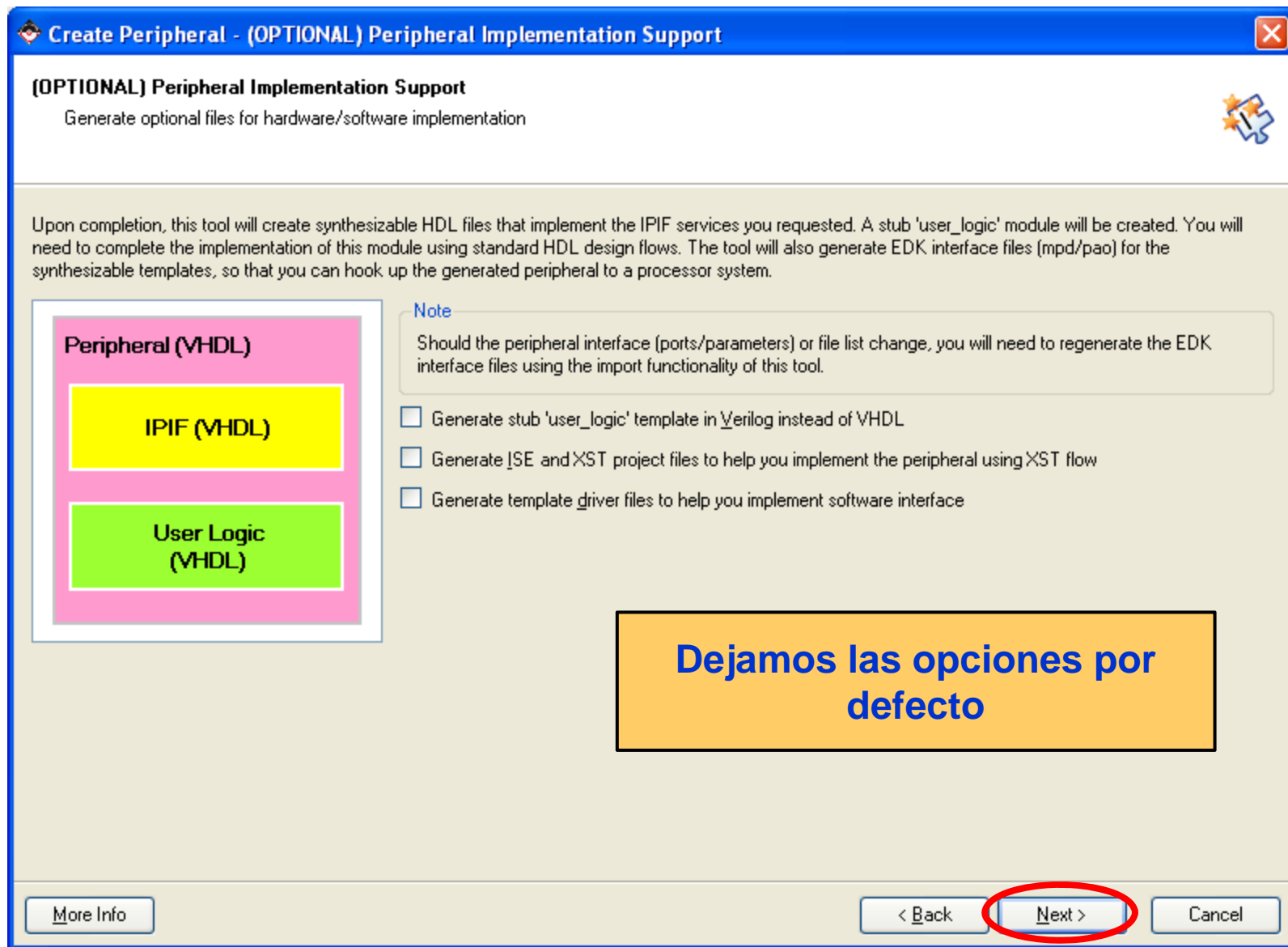
More Info < Back **Next >** Cancel



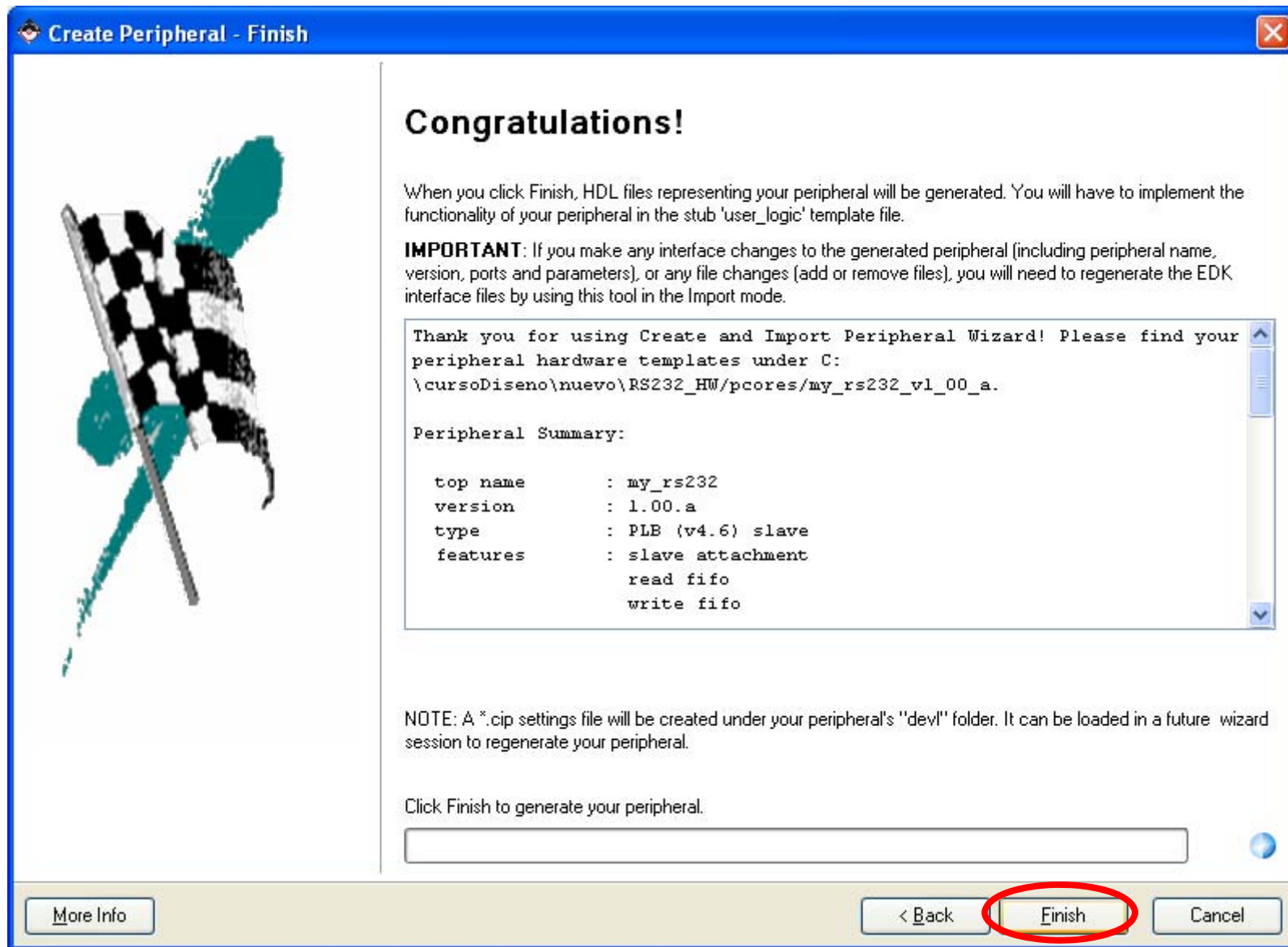
Creando el periférico con bus PLB



Creando el periférico con bus PLB



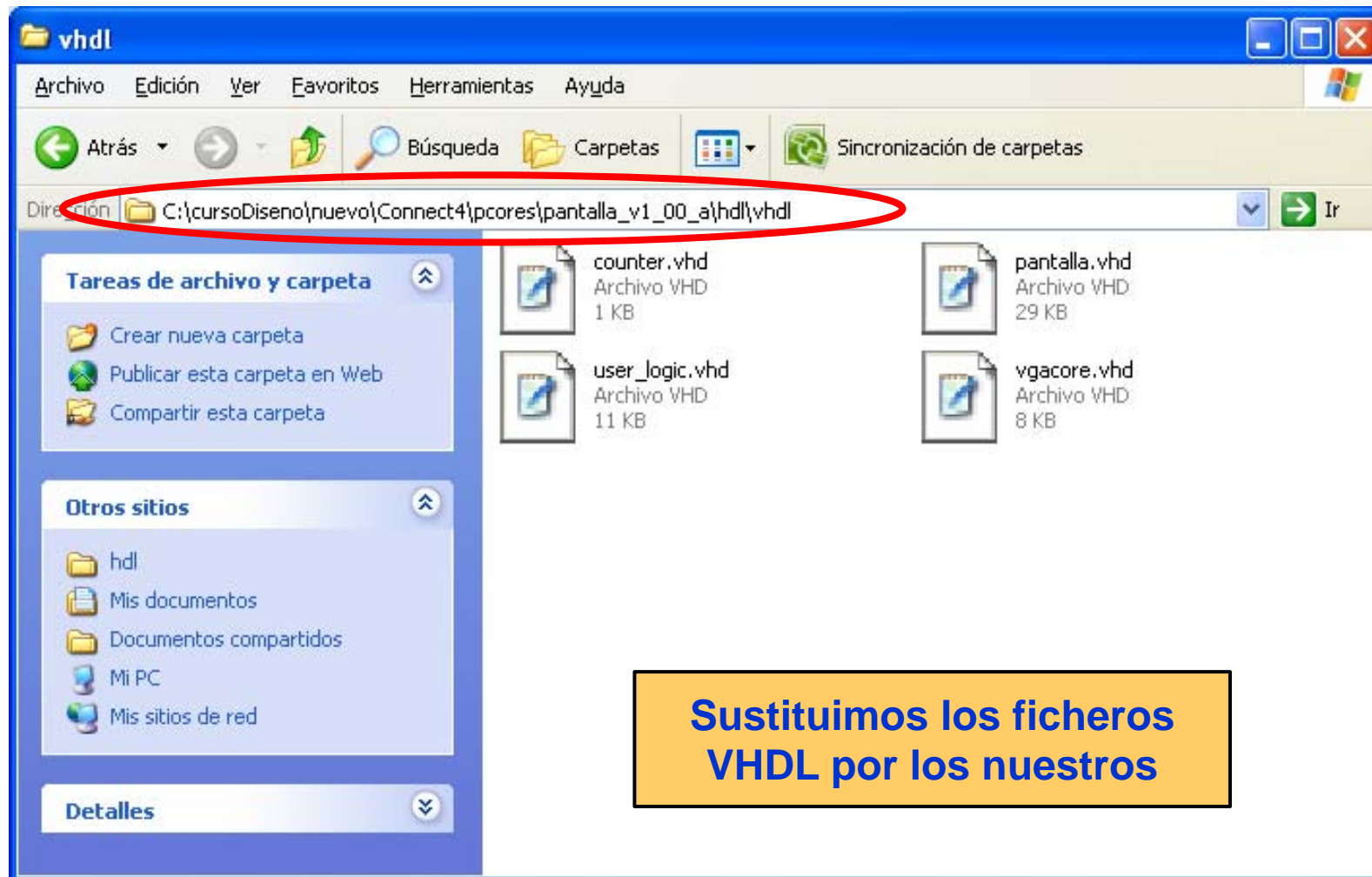
Creando el periférico con bus PLB



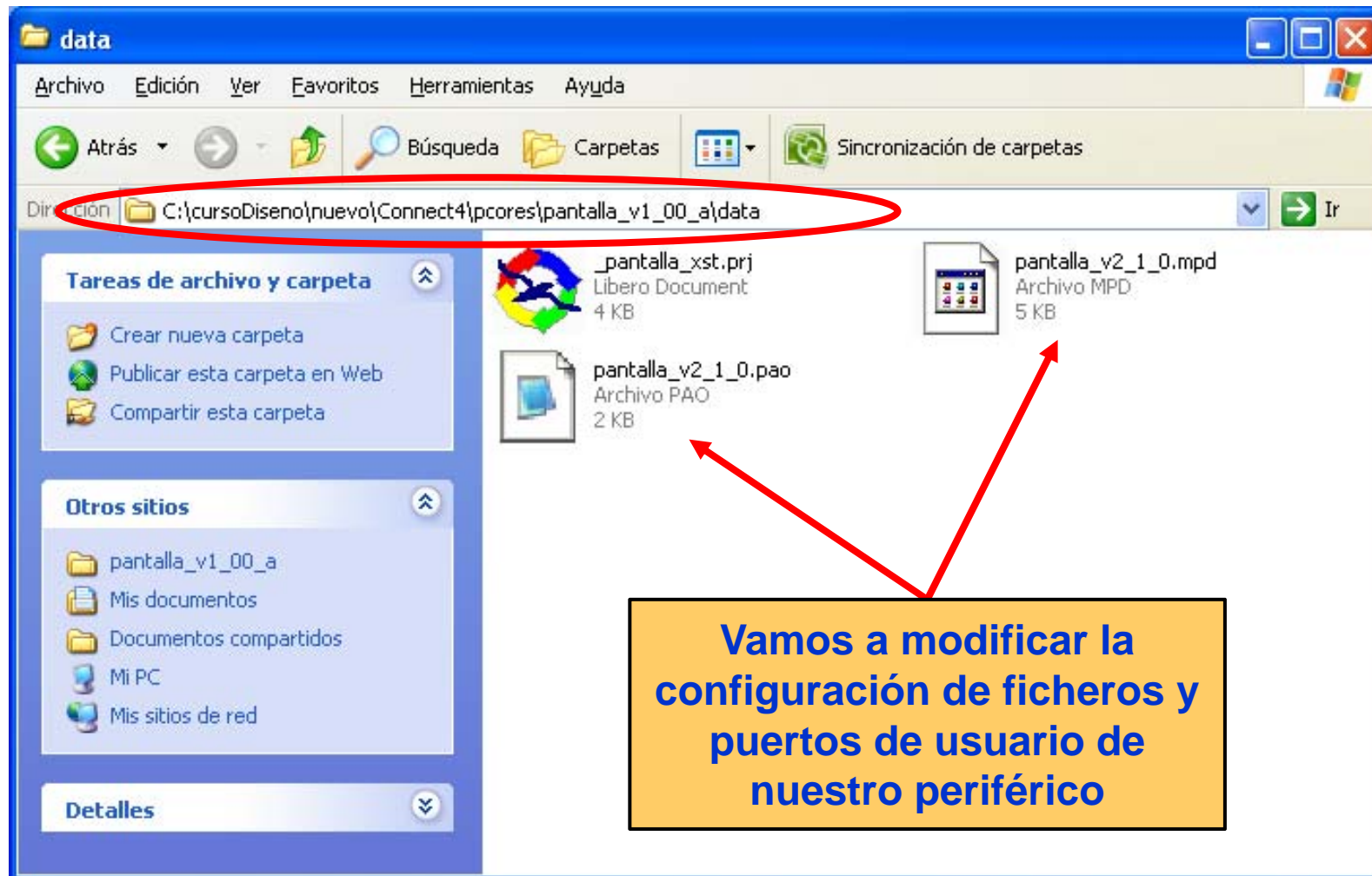
Hemos terminado de definir
la plantilla de nuestro
periférico



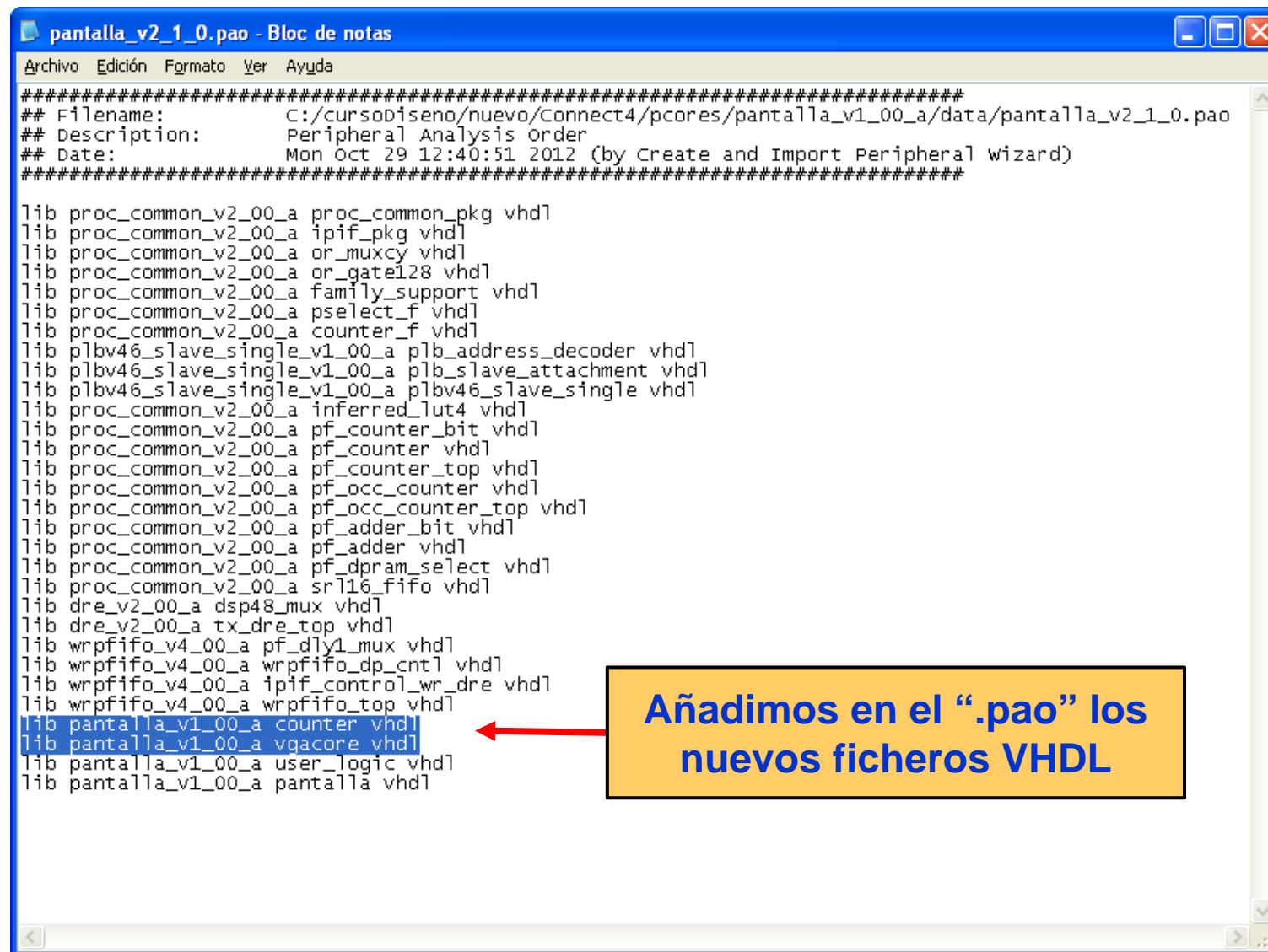
Modificando el periférico



Modificando el periférico



Modificando el periférico



```
#####
## Filename:      C:/cursos/biseno/nuevo/Connect4/pcoros/pantalla_v1_00_a/data/pantalla_v2_1_0.pao
## Description:   Peripheral Analysis Order
## Date:         Mon Oct 29 12:40:51 2012 (by Create and Import Peripheral wizard)
#####

lib proc_common_v2_00_a proc_common_pkg vhd1
lib proc_common_v2_00_a ipif_pkg vhd1
lib proc_common_v2_00_a or_muxcy vhd1
lib proc_common_v2_00_a or_gate128 vhd1
lib proc_common_v2_00_a family_support vhd1
lib proc_common_v2_00_a pselect_f vhd1
lib proc_common_v2_00_a counter_f vhd1
lib plbv46_slave_single_v1_00_a plb_address_decoder vhd1
lib plbv46_slave_single_v1_00_a plb_slave_attachment vhd1
lib plbv46_slave_single_v1_00_a plbv46_slave_single vhd1
lib proc_common_v2_00_a inferred_lut4 vhd1
lib proc_common_v2_00_a pf_counter_bit vhd1
lib proc_common_v2_00_a pf_counter vhd1
lib proc_common_v2_00_a pf_counter_top vhd1
lib proc_common_v2_00_a pf_occ_counter vhd1
lib proc_common_v2_00_a pf_occ_counter_top vhd1
lib proc_common_v2_00_a pf_adder_bit vhd1
lib proc_common_v2_00_a pf_adder vhd1
lib proc_common_v2_00_a pf_dpram_select vhd1
lib proc_common_v2_00_a srl16_fifo vhd1
lib dre_v2_00_a dsp48_mux vhd1
lib dre_v2_00_a tx_dre_top vhd1
lib wrpfifo_v4_00_a pf_dly1_mux vhd1
lib wrpfifo_v4_00_a wrpfifo_dp_cnt1 vhd1
lib wrpfifo_v4_00_a ipif_control_wr_dre vhd1
lib wrpfifo_v4_00_a wrpfifo_top vhd1
lib pantalla_v1_00_a counter vhd1
lib pantalla_v1_00_a vgacore vhd1
lib pantalla_v1_00_a user_logic vhd1
lib pantalla_v1_00_a pantalla vhd1
```

**Añadimos en el “.pao” los
nuevos ficheros VHDL**



Modificando el periférico

```
pantalla_v2_1_0.mpd - Bloc de notas
Archivo Edición Formato Ver Ayuda
PORT PLB_rdPrim = PLB_rdPrim, DIR = I, BUS = SPLB
PORT PLB_wrPrim = PLB_wrPrim, DIR = I, BUS = SPLB
PORT PLB_masterID = PLB_masterID, DIR = I, VEC = [0:(C_SPLB_MID_WIDTH-1)], BUS = SPLB
PORT PLB_abort = PLB_abort, DIR = I, BUS = SPLB
PORT PLB_busLock = PLB_busLock, DIR = I, BUS = SPLB
PORT PLB_RNW = PLB_RNW, DIR = I, BUS = SPLB
PORT PLB_BE = PLB_BE, DIR = I, VEC = [0:((C_SPLB_DWIDTH/8)-1)], BUS = SPLB
PORT PLB_Msize = PLB_Msize, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_size = PLB_size, DIR = I, VEC = [0:3], BUS = SPLB
PORT PLB_type = PLB_type, DIR = I, VEC = [0:2], BUS = SPLB
PORT PLB_lockErr = PLB_lockErr, DIR = I, BUS = SPLB
PORT PLB_wrDBus = PLB_wrDBus, DIR = I, VEC = [0:(C_SPLB_DWIDTH-1)], BUS = SPLB
PORT PLB_wrBurst = PLB_wrBurst, DIR = I, BUS = SPLB
PORT PLB_rdBurst = PLB_rdBurst, DIR = I, BUS = SPLB
PORT PLB_wrPendReq = PLB_wrPendReq, DIR = I, BUS = SPLB
PORT PLB_rdPendReq = PLB_rdPendReq, DIR = I, BUS = SPLB
PORT PLB_wrPendPri = PLB_wrPendPri, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_rdPendPri = PLB_rdPendPri, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_reqPri = PLB_reqPri, DIR = I, VEC = [0:1], BUS = SPLB
PORT PLB_TAttribute = PLB_TAttribute, DIR = I, VEC = [0:15], BUS = SPLB
PORT sl_addrAck = sl_addrAck, DIR = O, BUS = SPLB
PORT sl_ssize = sl_ssize, DIR = O, VEC = [0:1], BUS = SPLB
PORT sl_wait = sl_wait, DIR = O, BUS = SPLB
PORT sl_rearbitrate = sl_rearbitrate, DIR = O, BUS = SPLB
PORT sl_wrDack = sl_wrDack, DIR = O, BUS = SPLB
PORT sl_wrComp = sl_wrComp, DIR = O, BUS = SPLB
PORT sl_wrBTerm = sl_wrBTerm, DIR = O, BUS = SPLB
PORT sl_rdbus = sl_rdbus, DIR = O, VEC = [0:(C_SPLB_DWIDTH-1)], BUS = SPLB
PORT sl_rdwAddr = sl_rdwAddr, DIR = O, VEC = [0:3], BUS = SPLB
PORT sl_rdBack = sl_rdBack, DIR = O, BUS = SPLB
PORT sl_rdBComp = sl_rdBComp, DIR = O, BUS = SPLB
PORT sl_rdBTerm = sl_rdBTerm, DIR = O, BUS = SPLB
PORT sl_MBusy = sl_MBusy, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
PORT sl_MWrErr = sl_MWrErr, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
PORT sl_MRdErr = sl_MRdErr, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
PORT sl_MIRQ = sl_MIRQ, DIR = O, VEC = [0:(C_SPLB_NUM_MASTERS-1)], BUS = SPLB
PORT hsyncb = "", DIR = O
PORT vsyncb = "", DIR = O
PORT rgb = "", DIR = O, VEC = [0:8]
END
```

Añadimos en el “.mpd” los puertos de usuario de nuestro periférico



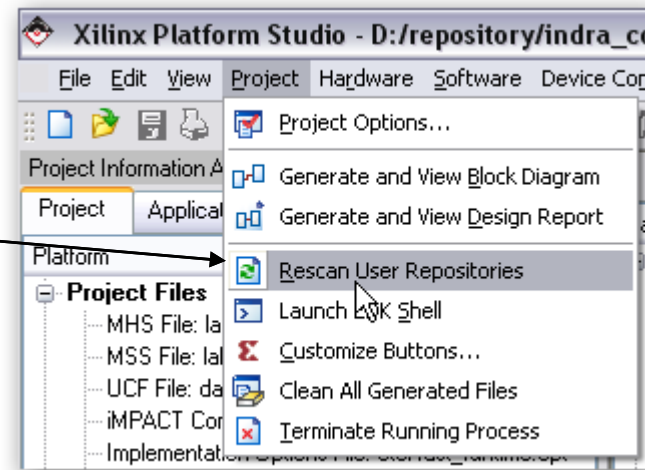
Hemos terminado de definir
el comportamiento de
nuestro periférico



Utilización de la aplicación Create Peripheral Wizard

- Abrir el proyecto EDK

Si EDK estaba ya abierto seleccionar :
Project -> Rescan User Repositories
Para actualizar los datos



Añadiendo el periférico PLB creado al diseño

El periférico está accesible dentro de la categoría "USER"

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.a
lmb_v10		lmb_v10	1.00.a
lmb_v10		lmb_v10	1.00.a
plb_v46		plb_v46	1.02.a
lmb_bram_if_cntlr		lmb_bram_if_cntlr	2.10.a
lmb_bram_if_cntlr		lmb_bram_if_cntlr	2.10.a
bram_block		bram_block	1.00.a
mdm		mdm	1.00.b
xps_gpio		xps_gpio	1.00.a
xps_gpio		xps_gpio	1.00.a
xps_uartlite		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.00.a
proc_sys_reset_0		proc_sys_reset	2.00.a



Añadiendo el periférico PLB creado al diseño

The screenshot shows the Xilinx Platform Studio interface with the 'System Assembly View' selected. The 'IP Catalog' on the left lists various components. The 'Bus Interfaces' table on the right shows the current configuration of the design. A red arrow points to the 'xps_gpio_SWITCHES' entry in the table, indicating its selection for addition to the design.

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	7.10.a
dlmb		lmb_v10	1.00.a
ilmb		lmb_v10	1.00.a
mb_plb		plb_v46	1.02.a
dlmb_cntlr		lmb_bram_if_cntlr	2.10.a
ilmb_cntlr		lmb_bram_if_cntlr	2.10.a
lmb_bram		bram_block	1.00.a
debug_module		mdm	1.00.b
pantalla_0		pantalla	1.00.a
SPLB	mb_plb		
xps_gpio_LEDS		xps_gpio	1.00.a
xps_gpio_SWITCHES		xps_gpio	1.00.a
RS232		xps_uartlite	1.00.a
clock_generator_0		clock_generator	2.00.a
proc_sys_reset_0		proc_sys_reset	2.00.a

Añadimos el periférico y lo conectamos al bus PLB como esclavo



Definiendo los puertos de usuario

Project Information Area

Project: Applications IP Catalog

Description: IP Version

EDK Install -- C:\Xilinx\10.1\EDK...

Project Local pcores -- C:\cursoDis...

USER

PANTALLA 1.00.a

Ports

Name	Net	Direction	Range
External Ports			
pantalla_0_hsyncb_pin	pantalla_0_hsyncb	0	
pantalla_0_vsyncb_pin	pantalla_0_vsyncb	0	
pantalla_0_rgb_pin	pantalla_0_rgb	0	[8:0]
xps_gpio_SWITCHES_GPIO_I...	xps_gpio_SWITCHES_GPIO	IO	[0:3]
xps_gpio_LEDS_GPIO_IO_pin	xps_gpio_LEDS_GPIO_IO	IO	[0:3]
sys_rst_pin	sys_rst_s	I	
sys_clk_pin	dcu_clk_s	I	
fpga_0_RS232_TX_pin	fpga_0_RS232_TX	0	
fpga_0_RS232_RX_pin	fpga_0_RS232_RX	I	
microblaze_0			
dmb			
ilmb			
mb_plb			
dmb_cntlr			
ilmb_cntlr			
lmb_bram			
debug_module			
pantalla_0			
rgb	pantalla_0_rgb	0	[8:0]
vsyncb	pantalla_0_vsyncb	0	
hsyncb	pantalla_0_hsyncb	0	
xps_gpio_LEDS			

Comprobamos que realmente aparezcan como puertos externos

Expandimos los puertos del componente y configuramos ambos como externos

Instance pantalla_0 port hsyncb connector undefined, using pantalla_0_hsyncb



Definiendo el espacio de direcciones del periférico

Xilinx Platform Studio - C:\cursoDiseno\nuevo\Connect4\lab1b.xmp - [System Assembly View1]

File Edit View Project Hardware Software Device Configuration Debug Simulation Window Help

Project Information Area

Project Applications IP Catalog

Description IP Version

- EDK Install -- C:\Xilinx\10.1\EDK...
- Analog
- Bus and Bridge
- Clock, Reset and Interrupt
- Communication High-Speed
- Communication Low-Speed
- DMA and Timer
- Debug
- General Purpose IO
- Interprocessor Communication
- Memory and Memory Controller
- PCI
- Peripheral Controller
- Processor
- Utility
- Project Local pcors -- C:\cursoDis...
- USER
 - PANTALLA 1.00.a

Bus Interfaces Ports Addresses

Generate Addresses

Instance	Name	Base Address	High Address	Size	Bus Interface
dlmb_cntrl	C_BASEADDR	0x00000000	0x00007fff	32K	SLMB
ilmb_cntrl	C_BASEADDR	0x00000000	0x00007fff	32K	SLMB
debug_module	C_BASEADDR	0x84400000	0x8440ffff	64K	SPLB
pantalla_0	C_BASEADDR	0xcd400000	0xcd40ffff	64K	SPLB
xps_gpio_LEDS	C_BASEADDR	0xcd414000	0xcd4141ff	512	SPLB
xps_gpio_SWITCHES	C_BASEADDR	0xcd418000	0xcd4181ff	512	SPLB
RS232	C_BASEADDR	0x84000000	0x8400ffff	64K	SPLB

Pulsamos para generar el espacio de direcciones. Por defecto, le asignará a nuestro periférico un espacio de 64K

System Assembly View Block Diagram

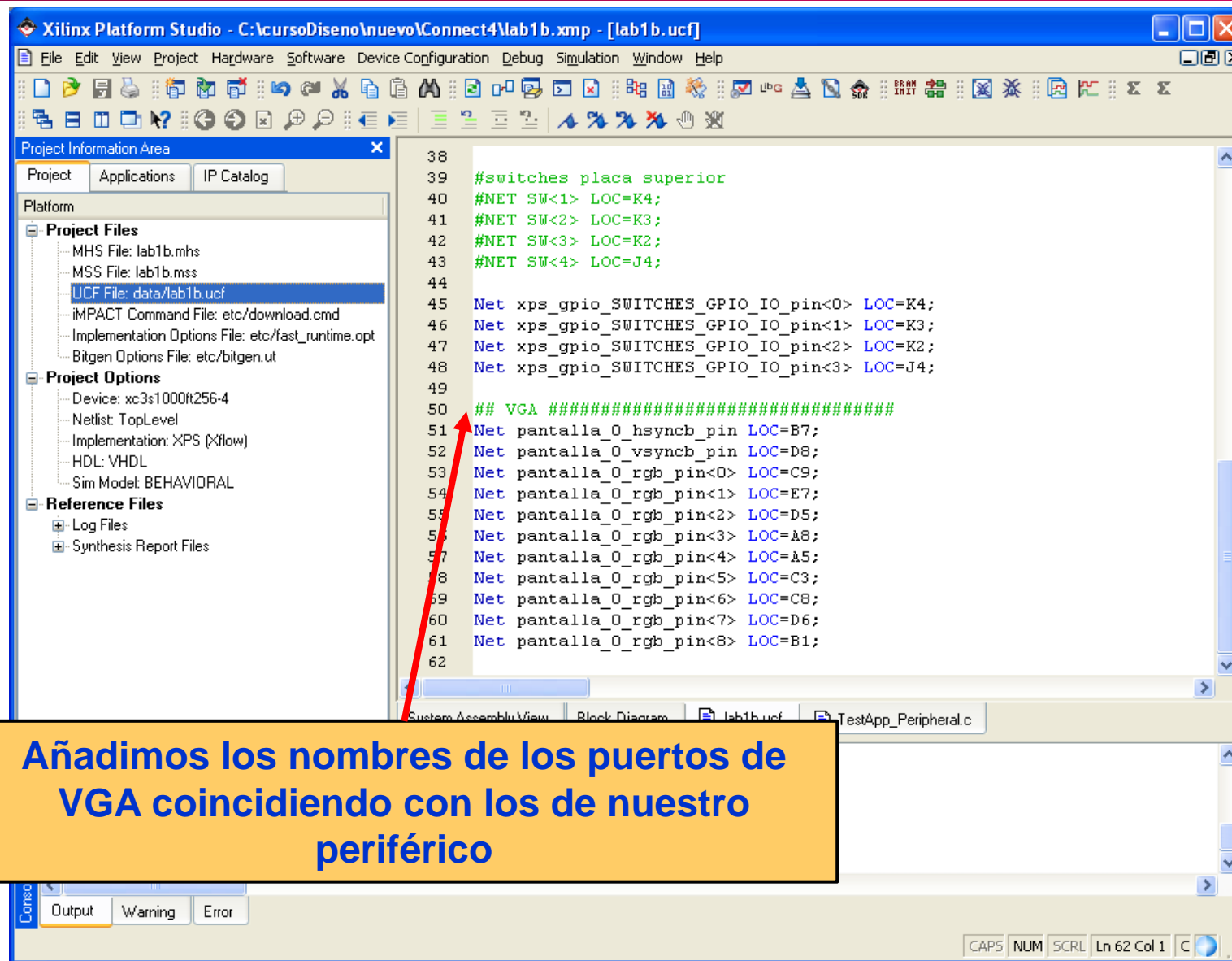
Console Window

```
(0x84000000-0x8400ffff) RS232 mb_plb
(0x84400000-0x8440ffff) debug_module mb_plb
(0xcd400000-0xcd40ffff) pantalla_0 mb_plb
(0xcd414000-0xcd4141ff) xps_gpio_LEDS mb_plb
(0xcd418000-0xcd4181ff) xps_gpio_SWITCHES mb_plb
```

Output Warning Error



Modificamos el fichero UCF



Project Information Area

- Project
- Applications
- IP Catalog

Platform

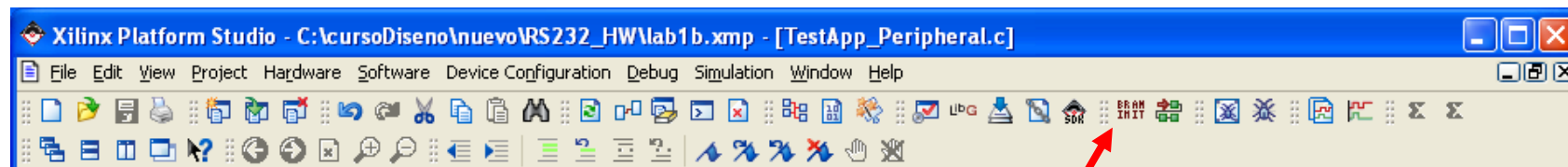
- Project Files
 - MHS File: lab1b.mhs
 - MSS File: lab1b.mss
 - UCF File: data/lab1b.ucf
 - iMPACT Command File: etc/download.cmd
 - Implementation Options File: etc/fast_runtime.opt
 - Bitgen Options File: etc/bitgen.ut
- Project Options
 - Device: xc3s1000r256-4
 - Netlist: TopLevel
 - Implementation: XPS (Xflow)
 - HDL: VHDL
 - Sim Model: BEHAVIORAL
- Reference Files
 - Log Files
 - Synthesis Report Files

```
38
39 #switches placa superior
40 #NET SW<1> LOC=K4;
41 #NET SW<2> LOC=K3;
42 #NET SW<3> LOC=K2;
43 #NET SW<4> LOC=J4;
44
45 Net xps_gpio_SWITCHES_GPIO_IO_pin<0> LOC=K4;
46 Net xps_gpio_SWITCHES_GPIO_IO_pin<1> LOC=K3;
47 Net xps_gpio_SWITCHES_GPIO_IO_pin<2> LOC=K2;
48 Net xps_gpio_SWITCHES_GPIO_IO_pin<3> LOC=J4;
49
50 ## VGA #####
51 Net pantalla_0_hsyncb_pin LOC=B7;
52 Net pantalla_0_vsyncb_pin LOC=D8;
53 Net pantalla_0_rgb_pin<0> LOC=C9;
54 Net pantalla_0_rgb_pin<1> LOC=E7;
55 Net pantalla_0_rgb_pin<2> LOC=D5;
56 Net pantalla_0_rgb_pin<3> LOC=A8;
57 Net pantalla_0_rgb_pin<4> LOC=A5;
58 Net pantalla_0_rgb_pin<5> LOC=C3;
59 Net pantalla_0_rgb_pin<6> LOC=C8;
60 Net pantalla_0_rgb_pin<7> LOC=D6;
61 Net pantalla_0_rgb_pin<8> LOC=B1;
62
```

Añadimos los nombres de los puertos de VGA coincidiendo con los de nuestro periférico



Generamos el .bit

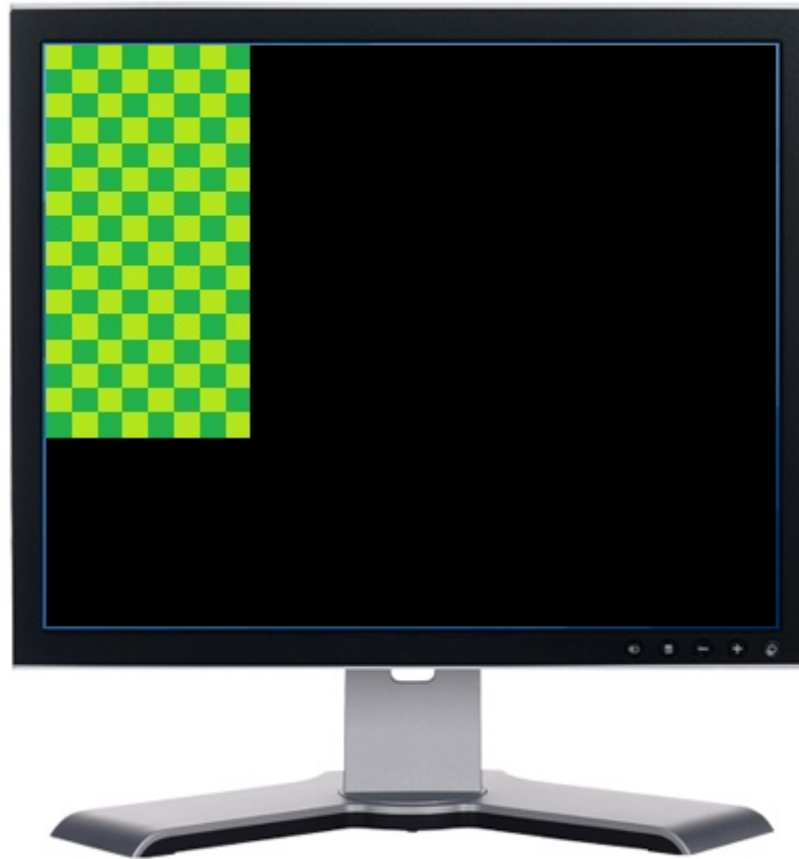


Generamos el fichero .bit



Resultado

- Al conectar el cable VGA a la placa de prototipado, debemos ver algo parecido a la imagen



Proyecto software

- Siempre que se genera un periférico dentro de un proyecto EDK, se genera un archivo *.h que tiene una biblioteca de funciones para leer y escribir en los registros y/o memorias del periférico.
- Dicho archivo se encuentra en \drivers\nombre_periferico\src
- Por ejemplo para escribir en la memoria fifo del periférico *pantalla* hay un archivo pantalla.h donde se define la función

```
#define PANTALLA_mWriteToFIFO(BaseAddress, DataOffset, Data) \  
    Xil_Out32((BaseAddress) + (PANTALLA_WRFIFO_DATA_OFFSET) + (DataOffset),  
    (Xuint32)(Data))
```

Donde DataOffset vale 0.

Además se generan 2 archivos *.c que permiten testear el periférico. Se debe copiar el archivo nombre_periferico.h en el directorio de nuestro proyecto software



Proyecto SW

C/C++ - peripheral_tests_0/src/pantalla.h - Xilinx SDK

File Edit Source Refactor Navigate Search Run Project Xilinx Tools Window Help

Project Explorer

- panalla2_hw_platform
 - peripheral_tests_0
 - Binaries
 - Includes
 - Debug
 - src

testperiph.c | pantalla.h | system.mss

```
* @return None.  
*  
* @note  
* C-style signature:  
* void PANTALLA_mResetWriteFIFO(Xuint32 BaseAddress)  
*/  
  
PANTALLA_mResetWriteFIFO(BaseAddress) \  
ut32 ((BaseAddress)+(PANTALLA_WRFIFO_RST_OFFSET), WRFIFO_RESET)  
  
status of PANTALLA write packet FIFO module.  
  
BaseAddress is the base address of the PANTALLA device.  
  
Status is the result of status checking.  
  
e signat  
PANTALLA  
32 PANTA  
  
PANTALLA  
In32 (Ba  
In32 (Ba  
  
32 bit data to PANTALLA write packet FIFO module.  
  
BaseAddress is the base address of the PANTALLA device.  
DataOffset is the offset from the data port to write to.  
Data is the value to be written to write packet FIFO.  
  
* @return None.  
*  
* @note  
* C-style signature:  
  
WRFIFO_FULL_MASK)  
  
PANTALLA_H  
xbasic_types.h  
xstatus.h  
xil_io.h  
# PANTALLA_USER_SLV  
# PANTALLA_WRFIFO_F  
# PANTALLA_WRFIFO_F  
# PANTALLA_WRFIFO_S  
# PANTALLA_WRFIFO_I  
# PANTALLA_WRFIFO_I  
# WRFIFO_FULL_MASK  
# WRFIFO_AF_MASK  
# WRFIFO_DL_MASK  
# WRFIFO_SCL_MASK  
# WRFIFO_WIDTH_MAS  
# WRFIFO_DREP_MASK  
# WRFIFO_VAC_MASK  
# WRFIFO_RESET  
# PANTALLA_mWriteRe  
# PANTALLA_mReadRe  
# PANTALLA_mResetWi  
# PANTALLA_mWriteFl  
# PANTALLA_mWriteFl  
# PANTALLA_mWriteTo  
# PANTALLA_SelfTest(v
```

Una vez copiado, pulsamos el botón derecho en src y seleccionamos Refresh

SDK Log

16:54:13 INFO : Processing command line option -hwspec C:\proyectos14\pantalla\EDK\SDK\SDK_Export\hw\system.xml.

C/C++ - peripheral_tests_0/src/pantalla.h - Xilinx SDK

pantalla2_hw_platform

- peripheral_tests_0
 - Binaries
 - Includes
 - Debug
 - src
 - gpio_header.h
 - pantalla_selftest.c
 - pantalla.c
 - pantalla.h
 - testperiph.c
 - uartlite_header.h
 - xgpio_tapp_example.c
 - xuartlite_selftest_example.c
 - Iscrip.ld
 - peripheral_tests_bsp_0

```

* @return None.
*
* @note
* C-style signature:
* void PANTALLA_mResetWriteFIFO(Xuint32 BaseAddress)
*/
#define PANTALLA_mResetWriteFIFO(BaseAddress) \
    Xil_Out32((BaseAddress)+(PANTALLA_WRFIFO_RST_OFFSET), WRFIFO_RESET)

/**
* Check status of PANTALLA write packet FIFO module.
*
* @param BaseAddress is the base address of the PANTALLA device.
*
* @return Status is the result of status checking.
*
* @note
* C-style signature:
* bool PANTALLA_mWriteFIFOFull(Xuint32 BaseAddress)
* Xuint32 PANTALLA_mWriteFIFOVacancy(Xuint32 BaseAddress)
*/
#define PANTALLA_mWriteFIFOFull(BaseAddress) \
    ((Xil_In32((BaseAddress)+(PANTALLA_WRFIFO_SR_OFFSET)) & WRFIFO_FULL_MASK) == WRFIFO_FULL_MASK)
#define PANTALLA_mWriteFIFOVacancy(BaseAddress) \
    ((Xil_In32((BaseAddress)+(PANTALLA_WRFIFO_SR_OFFSET)) & WRFIFO_VAC_MASK)

/**
* Write 32
*
* @param
* @param
* @param
*
* @return None.
*
* @note
* C-style signature:

```

Xilinx SDK

Xilinx SDK is based on Ecl

New to SDK?

You can get started by clicki
Or watch a 5 minute screen

Watch Now

Documentation

- Getting Started with
- EDK Concepts Tool
- Migrating from older
- Frequently asked qu

Known Issues

- Known issues in SC
- Xilinx Answer Recor

Questions, Comme

- Xilinx Forums
- Xilinx Technical Sup



Parte b1. Modificamos los colores de visualización

- En nuestro programa*.c incluimos el nuevo fichero *.h
#include "pantalla.h"
- Definimos los colores en nuestro programa *.c
 - ✓ #define ROJO 0x000001C0
 - ✓ #define VERDE 0x00000038
 - ✓ #define VERDE_OSCURO 0x00000018
- Recordamos: Los comandos tienen el siguiente formato de 32 bits
 - ✓ 7 LSB → Codifican el rectángulo a colorear
 - ✓ 9 MSB → Codifican el color para el rectángulo
 - ✓ El resto de bits no se utilizan
- Ejemplo de escritura en la fifo
 - posicion=columna *nfilas + fila; /* nfilas =16 */
 - color=ROJO;
 - Data= ((color & 0x1FF) << 23) | (posicion & 0x7f) ;
 - PANTALLA_mWriteToFIFO(baseaddr, 0, Data);
- Hacer un programa c que pida el color y posición de un rectángulo y lo modifique



Parte b2. Modificamos la posición de los rectángulos

- Modificar el hw para que la posición de los rectángulos que se visualizan pueda moverse a derecha e izquierda mediante 2 push buttons de la placa



Parte b3. Opcional. Modificamos el número de rectángulos a visualizar

- Modificar el hw para que el número de rectángulos (nfilas y ncolumnas) se determine por los switches. El máximo será 15*15.
- Modificar el hw para que el número de rectángulos se pase por memoria. Hacer un programa en *.c que pida el número de filas y columnas y se las pase al periférico

