

MÁSTER OFICIAL EN INGENIERÍA ELECTRÓNICA

SISTEMAS INTEGRADOS

BLOQUE 2: Herramienta de diseño para MicroBlaze: EDK

Curso 2013-2014

José Torres

Raimundo García

Julio Martos

Jesús Soret

Adrián Suárez

Pedro A. Martínez

Abraham Menéndez



SISTEMAS INTEGRADOS

Tema 2.- Herramienta de diseño para MicroBlaze: EDK

Máster Oficial en Ingeniería Electrónica
Curso 2013-14



Introducción al Diseño de Sistemas Integrados

Índice

- ❑ Herramienta Xilinx EDK.
 - ❑ Descripción hardware con EDK.
 - ❑ Síntesis e implementación en EDK.
-

Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- ❑ ¿Qué es EDK?
 - ❑ EDK (Embedded Development Kit) es la herramienta de diseño que Xilinx nos ofrece para realizar Sistemas Integrados en FPGAs, tanto para Procesadores Hardware (ARM, Power-PC) como para Procesadores Software (MicroBlaze)
 - ❑ Permite la integración de los componentes hardware del Sistema Integrado y del código software para que los mismos actúen. También permite la Simulación, Depuración y Programación de todo el diseño.



Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

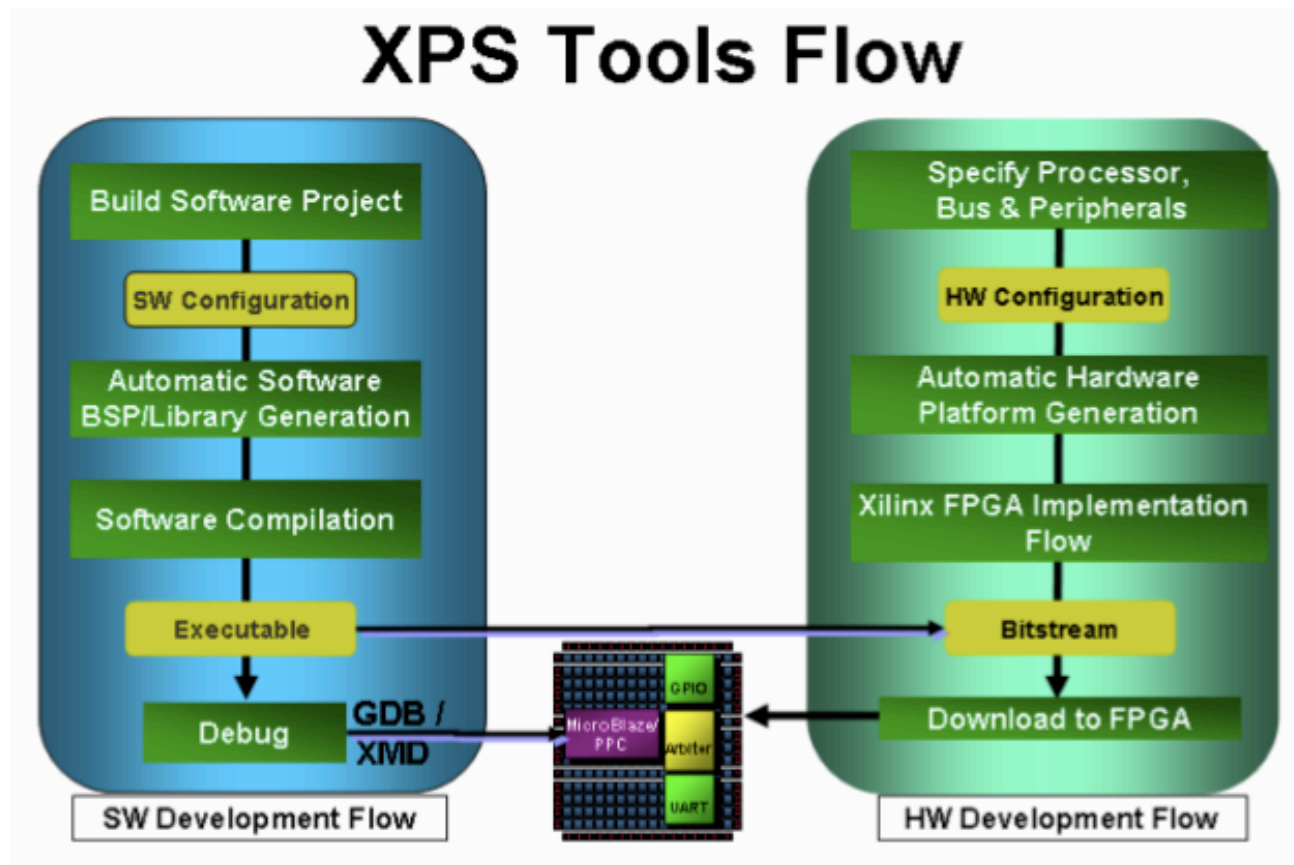
- ❑ Los programas que componen la herramienta EDK son:
 - ❑ Xilinx Platform Studio (XPS)
 - ❑ Software Development Kit (SDK)
 - ❑ Xilinx Microprocessor Debug (XMD)
 - ❑ ChipScope
 - ❑ ISE Foundation
 - ❑ ModelSim



Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

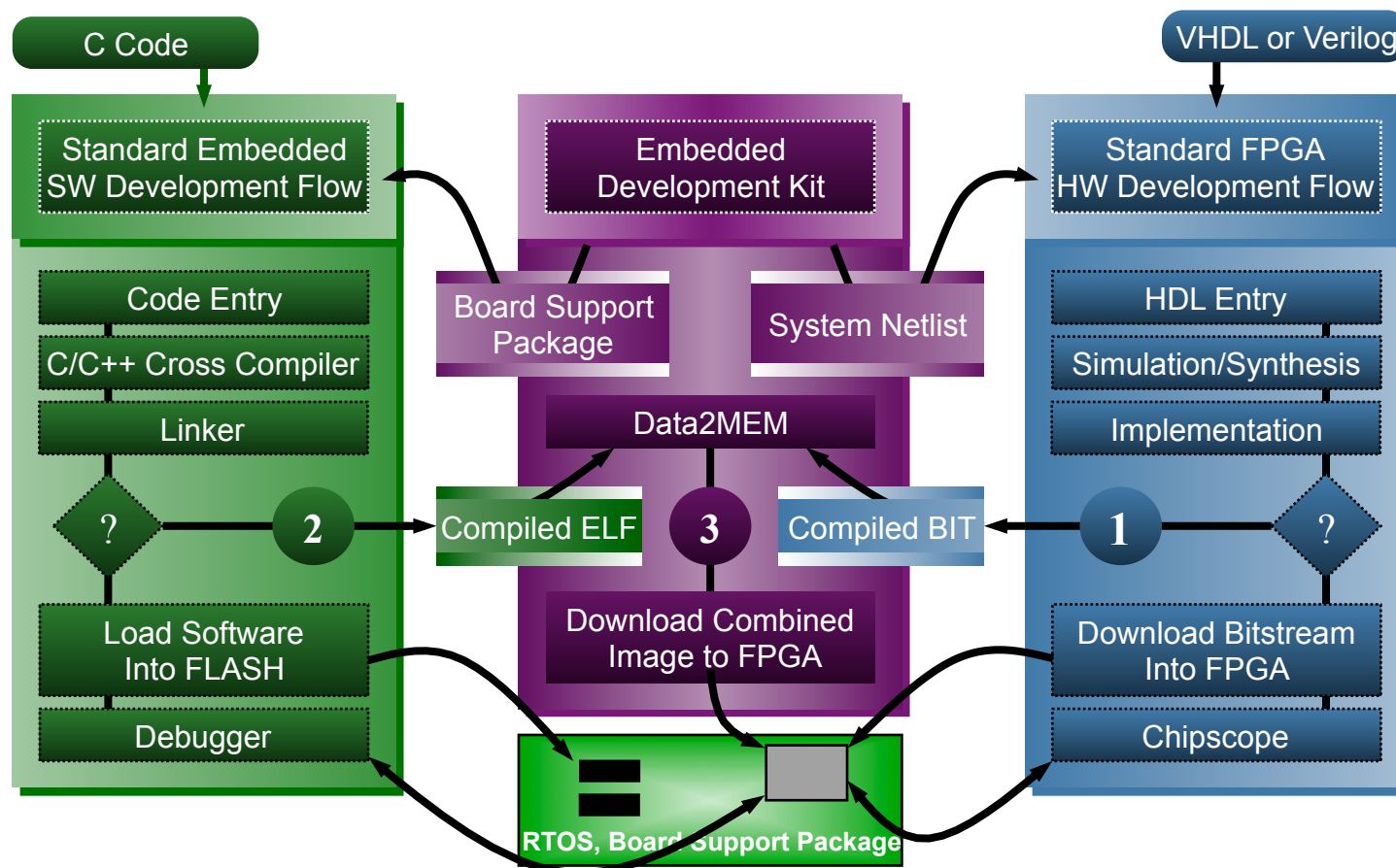
- Las fases básicas del proceso de diseño con EDK son:



Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

De forma más detallada:



Herramienta de diseño para MicroBlaze: EDK

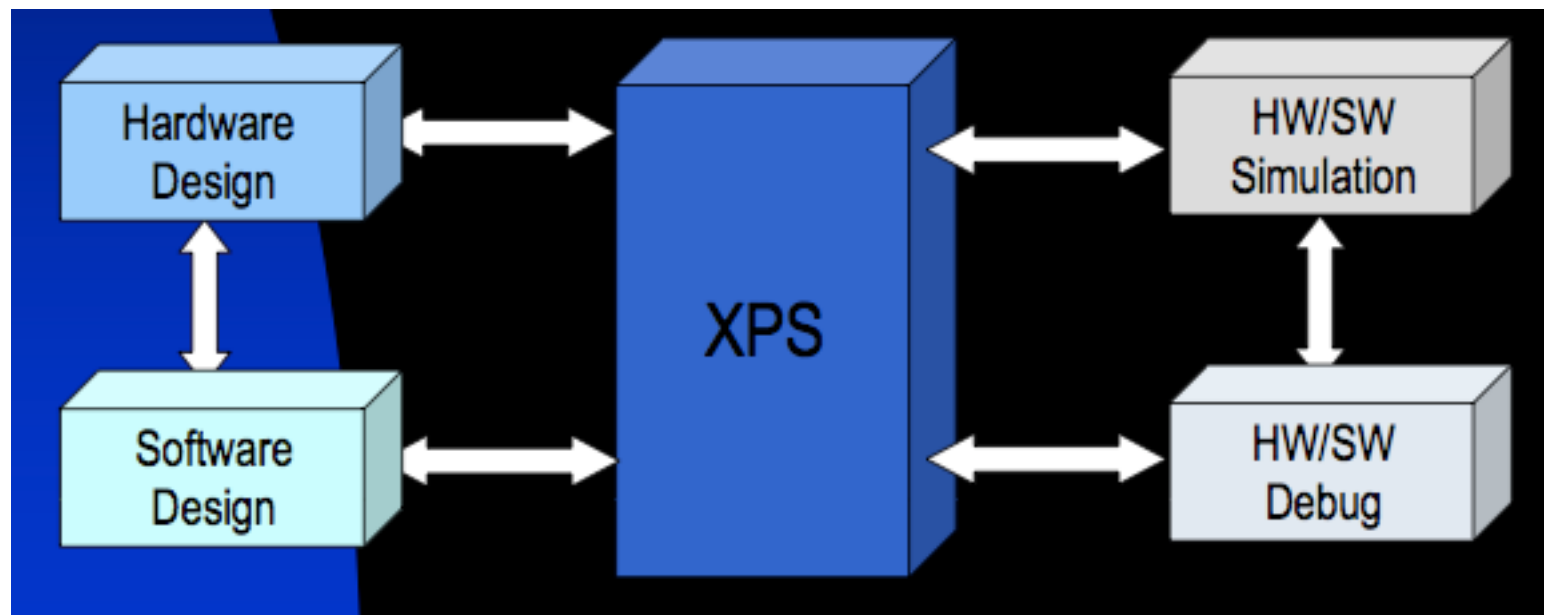
Herramienta Xilinx EDK

- ❑ Funciones del programa XPS:
 - ❑ Gestión de proyectos
 - ❑ Archivos Microprocessor Hardware Specification (MHS) y Microprocessor Software Specification (MSS)
 - ❑ Archivo principal Xilinx Microprocessor Project (XMP)
 - ❑ Gestión de aplicaciones software
 - ❑ Gestión de la plataforma
 - ❑ Ajustes del flujo de la herramienta.
 - ❑ Opciones de la plataforma hardware.
 - ❑ Llamada a distintas herramientas.
 - ❑ Depuración y simulación.
-

Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- ❑ Funciones del programa XPS:



Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- ❑ Ventana principal de XPS:

The screenshot displays the Xilinx Platform Studio (XPS) interface. The window title is "Xilinx Platform Studio - C:/User/Xilinx_2006/Curso_Microblaze_2007/Ejemplos_curso/Ejemplo_basico_RS232_Microblaze_Spartan_3E/system_xmp [System Assembly View1]". The interface is divided into several sections:

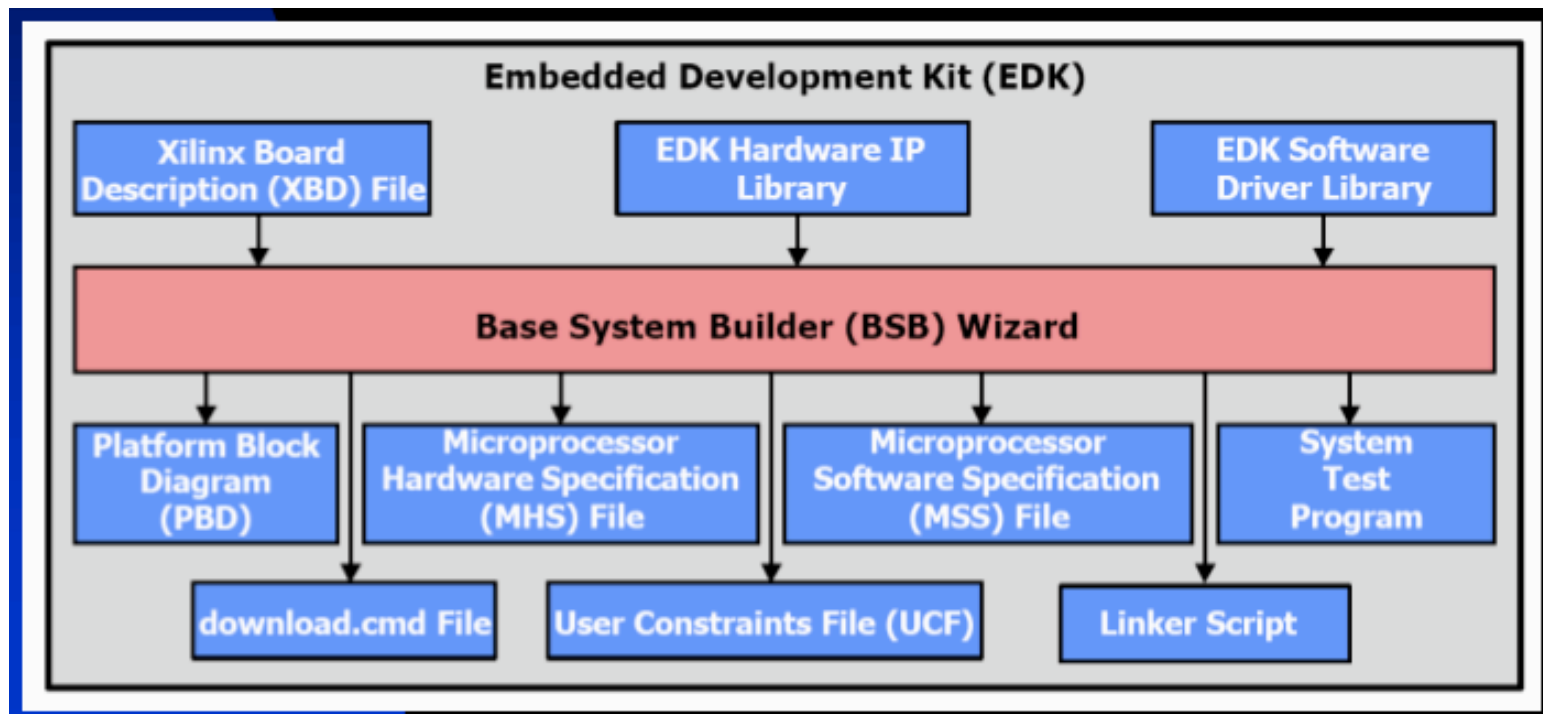
- Project Information Area:** Located on the left, it contains tabs for "IP Catalog", "Project", and "Applications". Under the "Project" tab, there are sections for "Project Files", "Project Options", and "Reference Files".
- Area de información del proyecto:** A blue-bordered box pointing to the Project Information Area.
- Area de conexiones del sistema empotrado:** A blue-bordered box pointing to the central "Connections" window, which shows a table of components and their connections.
- Area de consola:** A blue-bordered box pointing to the "Console View" at the bottom of the window, which includes "Output", "Warnings", and "Errors" tabs.

Name	Bus Connection	IP Type	IP Version
microblaze_0		microblaze	5.00 a
mb_spb		spb_v20	1.10 c
mb		mb_v10	1.00 a
mb		mb_v10	1.00 a
debug_module		spb_mdm	2.00 a
mb_cmb		mb_bram_f_cmb	2.00 a
mb_cmb		mb_bram_f_cmb	2.00 a
RS232_DCE		spb_serfile	1.00 b
LEDs_8bit		spb_gpio	3.01 b
DIP_Switches_4bit		spb_gpio	3.01 b
spb_timer_1		spb_timer	1.00 b
spb_timer_2		spb_timer	1.00 b
spb_irq_0		spb_irq	1.00 c
mb_bram		bram_block	1.00 a
dcm_0		dcm_module	1.00 a

Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

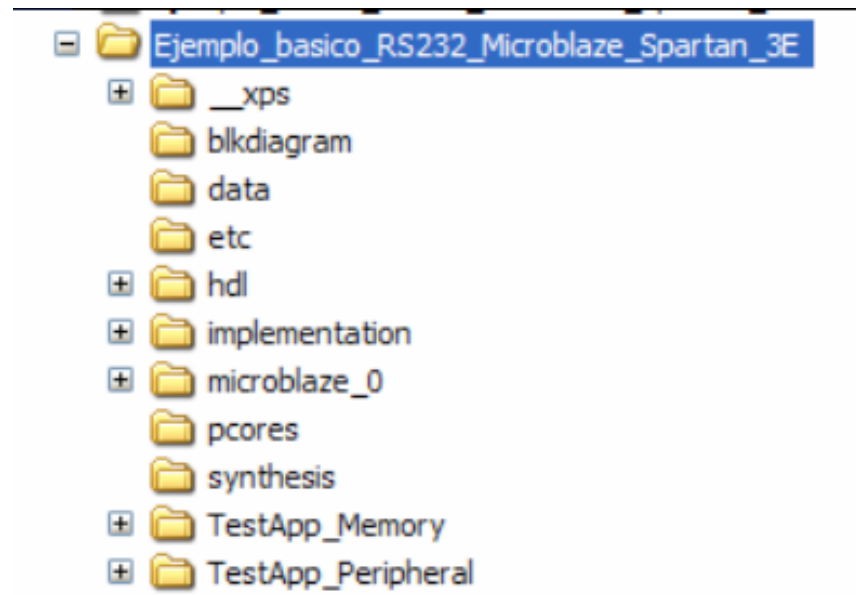
- La utilidad Base System Builder (BSB) incluida en XPS nos ayudará a crear una plataforma hardware y unas aplicaciones software de test para nuestro sistema empotrado.



Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

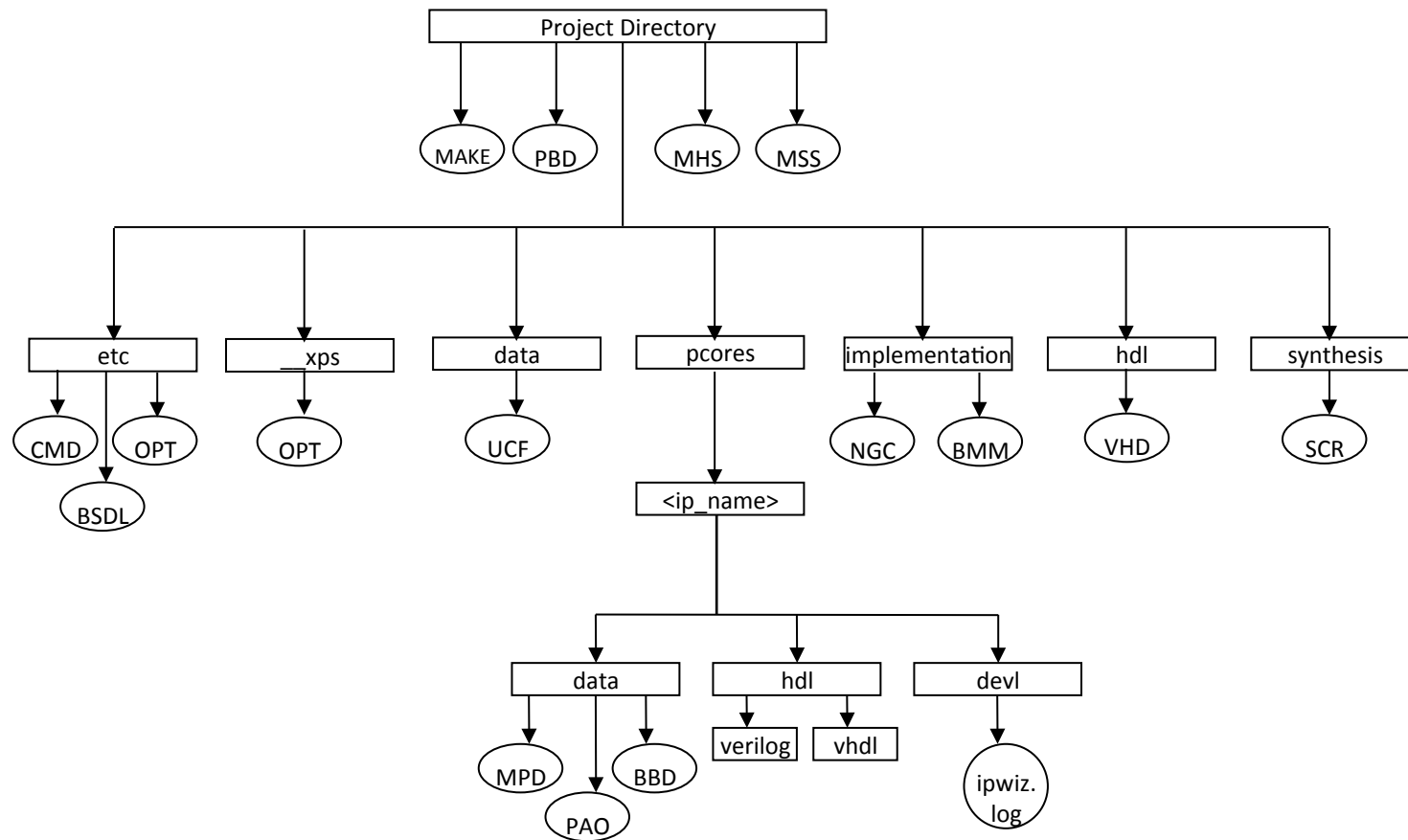
- ❑ Directorios generados por XPS:
 - ❑ Los directorios implementation, synthesis y pcores almacenan la información del hardware del sistema.
 - ❑ Los directorios TestAppMemory y TestAppPeripheral corresponden a dos aplicaciones software para testear el correcto funcionamiento del hardware.



Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- Archivos generados por XPS:

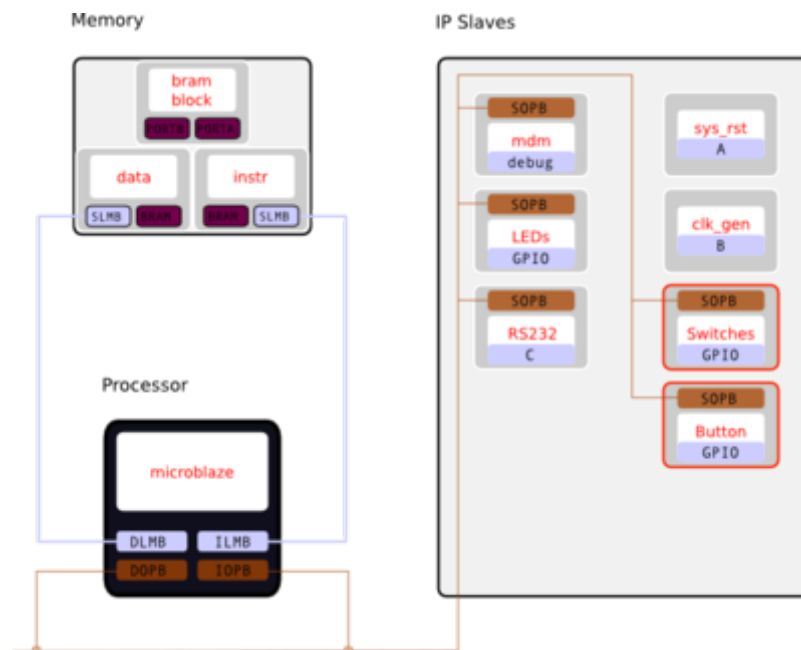


Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

❑ Ejemplo de archivo XMP (Xilinx Microprocessor Project)

- ❑ Descripción de las diferentes opciones del proyecto, tanto hardware como software.
- ❑ En general no debe modificarse aunque algunos errores de la herramienta pueden obligar a hacerlo.



```
LX9_AXI_ACL_system.xmp
#Please do not modify this file by hand
XmpVersion: 13.2
VerMgmt: 13.2
IntStyle: default
Flow: ise
MHS File: LX9_AXI_ACL_system.mhs
Architecture: spartan6
Device: xc6slx9
Package: csg324
SpeedGrade: -2
UserCmd1:
UserCmd1Type: 0
UserCmd2:
UserCmd2Type: 0
GenSimTB: 1
SdkExportBmmBit: 1
SdkExportDir: SDK/SDK_Export
InsertNoPads: 0
WarnForEAArch: 1
HdlLang: Verilog
SimModel: BEHAVIORAL
ExternalMemSim: 0
UcfFile: data/LX9_AXI_ACL_system.ucf
EnablePartTimingError: 1
ShowLicenseDialog: 1
ICacheAddr: MCB3_LPDDR,C_S0_AXI_BASEADDR
ICacheAddr: MCB3_LPDDR,C_S1_AXI_BASEADDR
ICacheAddr: MCB3_LPDDR,C_S2_AXI_BASEADDR
ICacheAddr: MCB3_LPDDR,C_S3_AXI_BASEADDR
ICacheAddr: MCB3_LPDDR,C_S4_AXI_BASEADDR
ICacheAddr: MCB3_LPDDR,C_S5_AXI_BASEADDR
DCacheAddr: MCB3_LPDDR,C_S0_AXI_BASEADDR
DCacheAddr: MCB3_LPDDR,C_S1_AXI_BASEADDR
DCacheAddr: MCB3_LPDDR,C_S2_AXI_BASEADDR
DCacheAddr: MCB3_LPDDR,C_S3_AXI_BASEADDR
DCacheAddr: MCB3_LPDDR,C_S4_AXI_BASEADDR
DCacheAddr: MCB3_LPDDR,C_S5_AXI_BASEADDR
Processor: microblaze_0
ElfImp:
ElfSim:
```

Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- Ejemplo de archivo MHS (Microprocessor Hardware Specification)

```
PORT fpga_0_RS232_RX_pin = fpga_0_RS232_RX, DIR = I
PORT fpga_0_RS232_TX_pin = fpga_0_RS232_TX, DIR = O
PORT fpga_0_LEDs_8Bit_GPIO_d_out_pin = fpga_0_LEDs_8Bit_GPIO_d_out, DIR = O, VEC = [0:7]
PORT fpga_0_LED_7SEGMENT_GPIO_d_out_pin = fpga_0_LED_7SEGMENT_GPIO_d_out, DIR = O, VEC = [0:11]
PORT fpga_0_Push_Buttons_3Bit_GPIO_in_pin = fpga_0_Push_Buttons_3Bit_GPIO_in, DIR = I, VEC = [0:2]
PORT fpga_0_DIP_Switches_8Bit_GPIO_in_pin = fpga_0_DIP_Switches_8Bit_GPIO_in, DIR = I, VEC = [0:7]
PORT fpga_0_SRAM_256Kx32_Mem_A_pin = fpga_0_SRAM_256Kx32_Mem_A, DIR = O, VEC = [12:29]
PORT fpga_0_SRAM_256Kx32_Mem_DQ_pin = fpga_0_SRAM_256Kx32_Mem_DQ, DIR = IO, VEC = [0:31]
PORT fpga_0_SRAM_256Kx32_Mem_CEN_pin = fpga_0_SRAM_256Kx32_Mem_CEN, DIR = O, VEC = [0:0]
PORT fpga_0_SRAM_256Kx32_Mem_CEN_1_pin = fpga_0_SRAM_256Kx32_Mem_CEN_1, DIR = O, VEC = [0:0]
PORT fpga_0_SRAM_256Kx32_Mem_WEN_pin = fpga_0_SRAM_256Kx32_Mem_WEN, DIR = O
PORT fpga_0_SRAM_256Kx32_Mem_BEN_pin = fpga_0_SRAM_256Kx32_Mem_BEN, DIR = O, VEC = [0:3]
PORT sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 50000000
PORT sys_rst_pin = sys_rst_s, DIR = I, RST_POLARITY = 1, SIGIS = RST

BEGIN microblaze
PARAMETER INSTANCE = microblaze_0
PARAMETER HW_VER = 5.00.a
PARAMETER C_USE_FPU = 0
PARAMETER C_DEBUG_ENABLED = 1
PARAMETER C_NUMBER_OF_PC_BRK = 2
BUS_INTERFACE DLMB = dlmb
BUS_INTERFACE ILMB = ilmb
BUS_INTERFACE DOPB = mb_opb
BUS_INTERFACE IOPB = mb_opb
PORT DBG_CAPTURE = DBG_CAPTURE_s
PORT DBG_CLK = DBG_CLK_s
PORT DBG_REG_EN = DBG_REG_EN_s
PORT DBG_TDI = DBG_TDI_s
PORT DBG_TDO = DBG_TDO_s
PORT DBG_UPDATE = DBG_UPDATE_s

BEGIN opb_v20
PARAMETER INSTANCE = mb_opb
PARAMETER HW_VER = 1.10.c
PARAMETER C_EXT_RESET_HIGH = 1
PORT SYS_Rst = sys_rst_s
PORT OPR_Clk = sys_clk_s
```

Descripción de los puertos (terminales externos de la FPGA)

Descripción de los distintos componentes del sistema empotrado

Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- Ejemplo de archivo MSS (Microprocessor Software Specification)

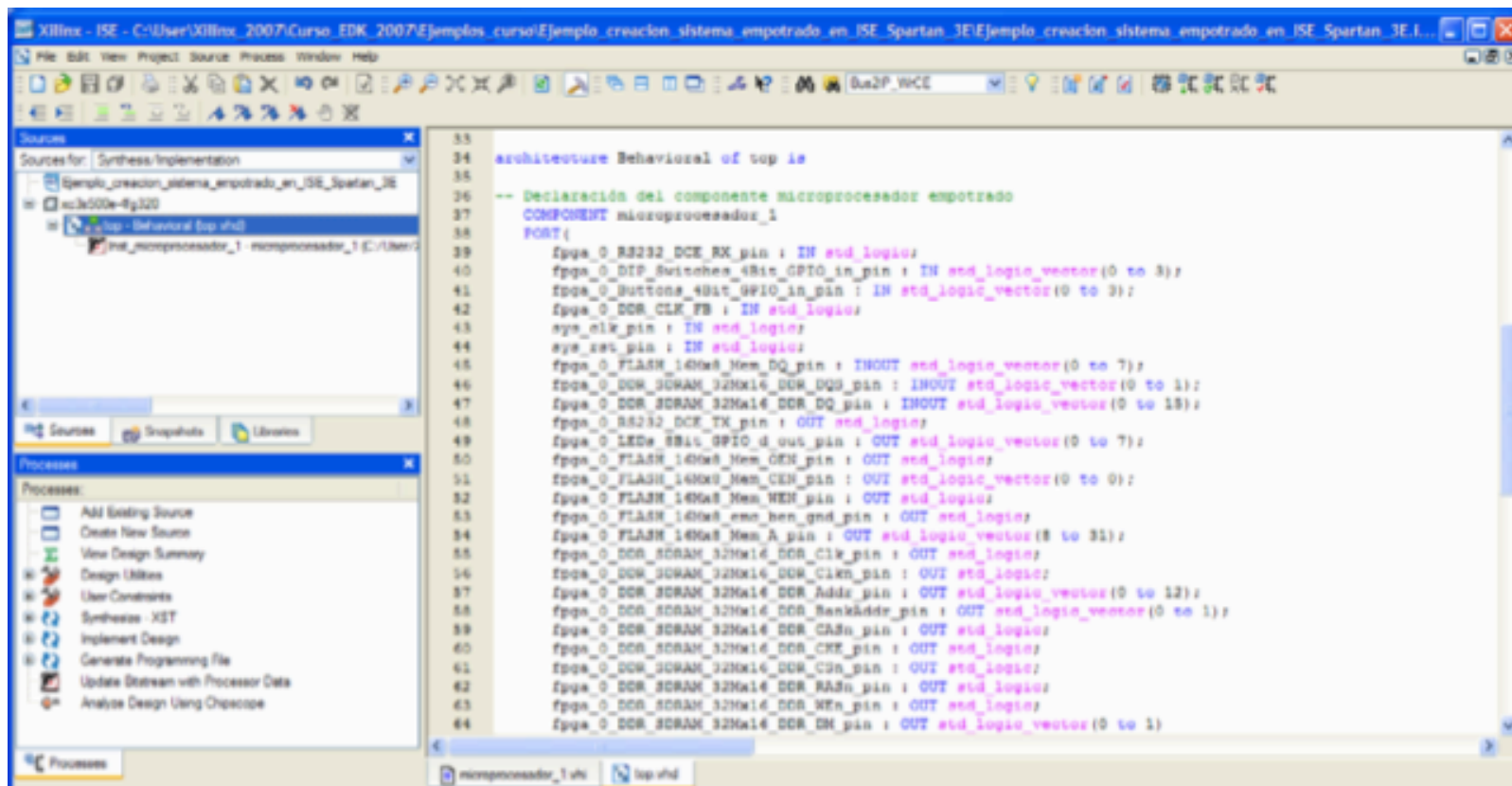
The image shows a snippet of an MSS file with four sections circled in red. Red arrows point from each section to a descriptive label on the right:

- Section 1:** `BEGIN OS`
`PARAMETER OS_NAME = standalone`
`PARAMETER OS_VER = 1.00.a`
`PARAMETER PROC_INSTANCE = microblaze_0`
`PARAMETER STDIN = RS232`
`PARAMETER STDOUT = RS232`
`END`
Annotation: Descripción del sistema operativo
- Section 2:** `BEGIN PROCESSOR`
`PARAMETER DRIVER_NAME = cpu`
`PARAMETER DRIVER_VER = 1.00.a`
`PARAMETER HW_INSTANCE = microblaze_0`
`PARAMETER COMPILER = mb-gcc`
`PARAMETER ARCHIVER = mb-ar`
`PARAMETER XMDSTUB_PERIPHERAL = debug_module`
`END`
Annotation: Descripción del “driver” de la CPU
- Section 3:** `BEGIN DRIVER`
`PARAMETER DRIVER_NAME = opbarb`
`PARAMETER DRIVER_VER = 1.02.a`
`PARAMETER HW_INSTANCE = mb_opb`
`END`
Annotation: Descripción de los “drivers” de los distintos periféricos
- Section 4:** `BEGIN DRIVER`
`PARAMETER DRIVER_NAME = uartlite`
`PARAMETER DRIVER_VER = 1.01.a`
`PARAMETER HW_INSTANCE = debug_module`
`END`
- Section 5:** `BEGIN DRIVER`
`PARAMETER DRIVER_NAME = bram`
`PARAMETER DRIVER_VER = 1.00.a`
`PARAMETER HW_INSTANCE = dlmb_cntlr`
`END`

Herramienta de diseño para MicroBlaze: EDK

Herramienta Xilinx EDK

- Los sistemas empujados diseñados mediante XPS pueden integrarse en un sistema más complejo mediante ISE Foundation.

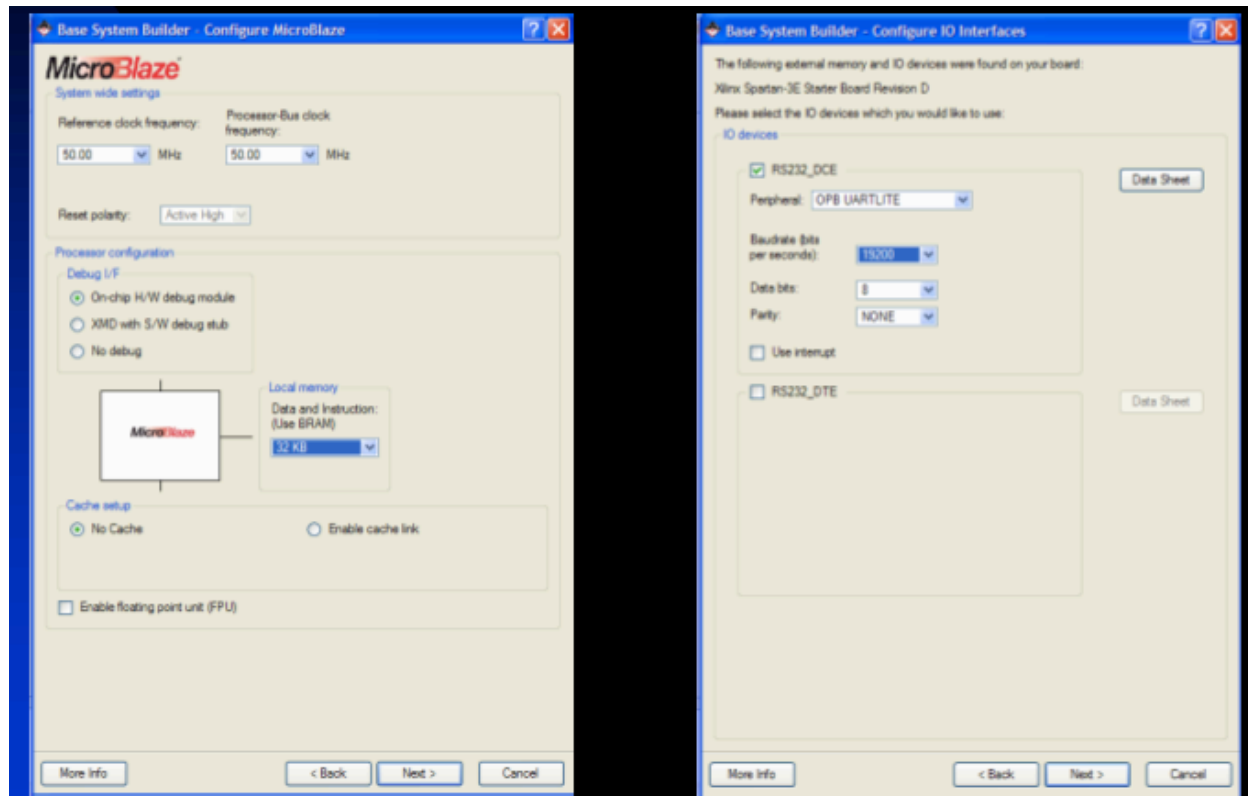


```
33
34 architecture Behavioral of top is
35
36 -- Declaración del componente microprocesador empujado
37 COMPONENT microprocesador_1
38   PORT(
39     fpga_0_RS232_DCE_RX_pin : IN std_logic;
40     fpga_0_DIP_Switches_4Bit_GPIO_in_pin : IN std_logic_vector(0 to 3);
41     fpga_0_Buttons_4Bit_GPIO_in_pin : IN std_logic_vector(0 to 3);
42     fpga_0_DCR_CLK_FB : IN std_logic;
43     eye_clk_pin : IN std_logic;
44     eye_reset_pin : IN std_logic;
45     fpga_0_FLASH_16Mx8_Mem_DQ_pin : INOUT std_logic_vector(0 to 7);
46     fpga_0_DCR_SDRAM_32Mx16_DCR_DQS_pin : INOUT std_logic_vector(0 to 1);
47     fpga_0_DCR_SDRAM_32Mx16_DCR_DQ_pin : INOUT std_logic_vector(0 to 15);
48     fpga_0_RS232_DCE_TX_pin : OUT std_logic;
49     fpga_0_LEDS_8Bit_GPIO_d_out_pin : OUT std_logic_vector(0 to 7);
50     fpga_0_FLASH_16Mx8_Mem_CEN_pin : OUT std_logic;
51     fpga_0_FLASH_16Mx8_Mem_CEN_pin : OUT std_logic_vector(0 to 0);
52     fpga_0_FLASH_16Mx8_Mem_WEN_pin : OUT std_logic;
53     fpga_0_FLASH_16Mx8_mem_ben_gnd_pin : OUT std_logic;
54     fpga_0_FLASH_16Mx8_Mem_A_pin : OUT std_logic_vector(8 to 31);
55     fpga_0_DCR_SDRAM_32Mx16_DCR_Clk_pin : OUT std_logic;
56     fpga_0_DCR_SDRAM_32Mx16_DCR_Clkn_pin : OUT std_logic;
57     fpga_0_DCR_SDRAM_32Mx16_DCR_Adda_pin : OUT std_logic_vector(0 to 12);
58     fpga_0_DCR_SDRAM_32Mx16_DCR_BankAddr_pin : OUT std_logic_vector(0 to 1);
59     fpga_0_DCR_SDRAM_32Mx16_DCR_CASn_pin : OUT std_logic;
60     fpga_0_DCR_SDRAM_32Mx16_DCR_CKE_pin : OUT std_logic;
61     fpga_0_DCR_SDRAM_32Mx16_DCR_CSn_pin : OUT std_logic;
62     fpga_0_DCR_SDRAM_32Mx16_DCR_RASn_pin : OUT std_logic;
63     fpga_0_DCR_SDRAM_32Mx16_DCR_WEn_pin : OUT std_logic;
64     fpga_0_DCR_SDRAM_32Mx16_DCR_DM_pin : OUT std_logic_vector(0 to 1)
```


Herramienta de diseño para MicroBlaze: EDK

Descripción hardware EDK

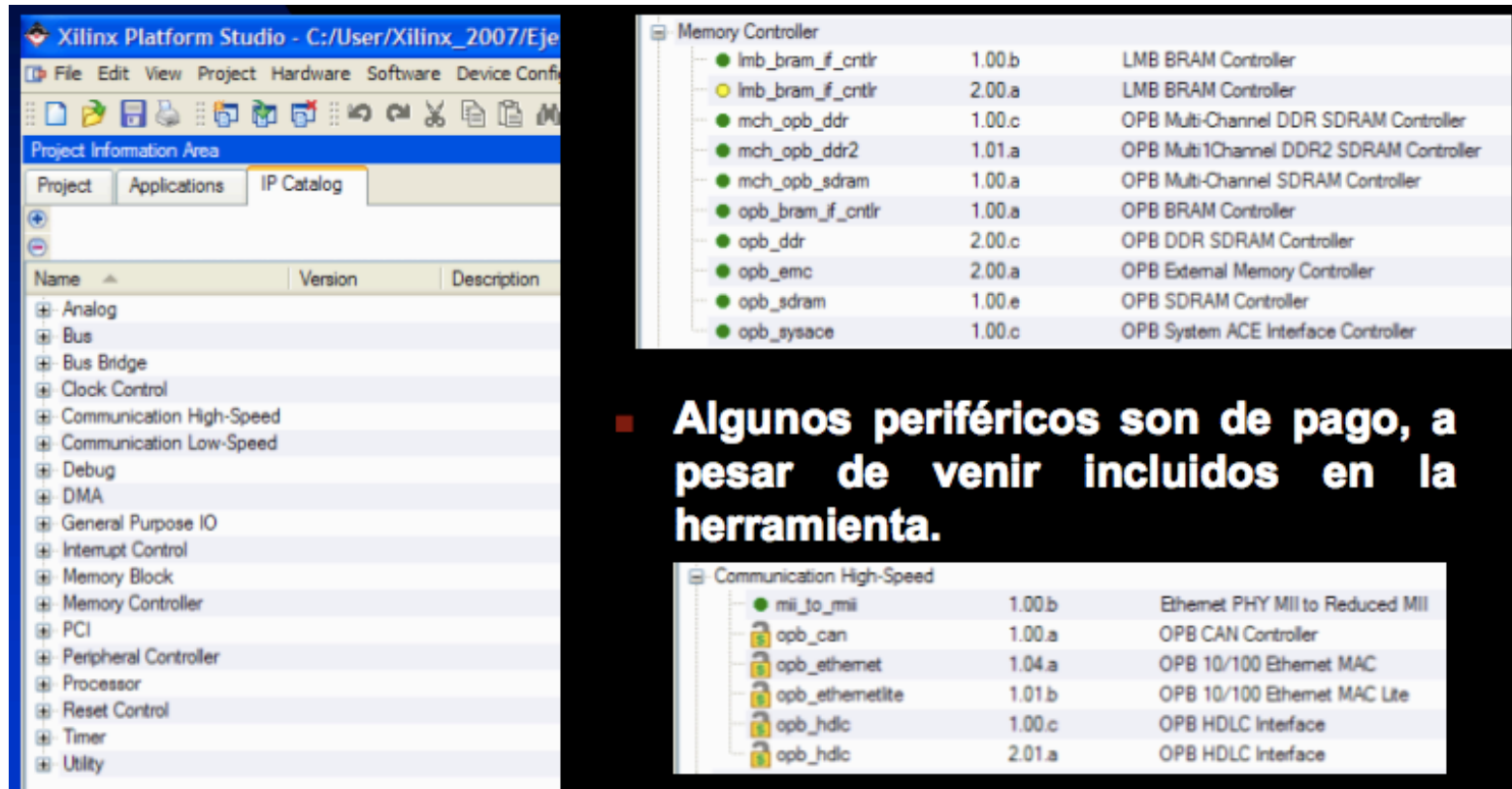
- Mediante la herramienta BSB definiremos las opciones del hardware (Microblaze) y añadiremos los periféricos de nuestro sistema empotrado.



Herramienta de diseño para MicroBlaze: EDK

Descripción hardware EDK

- Posteriormente podremos añadir alguno de los periféricos predefinidos en EDK.



The screenshot shows the Xilinx Platform Studio interface. On the left, the IP Catalog is visible, listing various hardware blocks. On the right, two expanded views of IP blocks are shown, each with a table of details.

Memory Controller

Name	Version	Description
lmb_bram_if_cntlr	1.00.b	LMB BRAM Controller
lmb_bram_if_cntlr	2.00.a	LMB BRAM Controller
mch_opb_dds	1.00.c	OPB Multi-Channel DDR SDRAM Controller
mch_opb_dds2	1.01.a	OPB Multi-Channel DDR2 SDRAM Controller
mch_opb_sdrsm	1.00.a	OPB Multi-Channel SDRAM Controller
opb_bram_if_cntlr	1.00.a	OPB BRAM Controller
opb_dds	2.00.c	OPB DDR SDRAM Controller
opb_emc	2.00.a	OPB External Memory Controller
opb_sdrsm	1.00.e	OPB SDRAM Controller
opb_sysace	1.00.c	OPB System ACE Interface Controller

Communication High-Speed

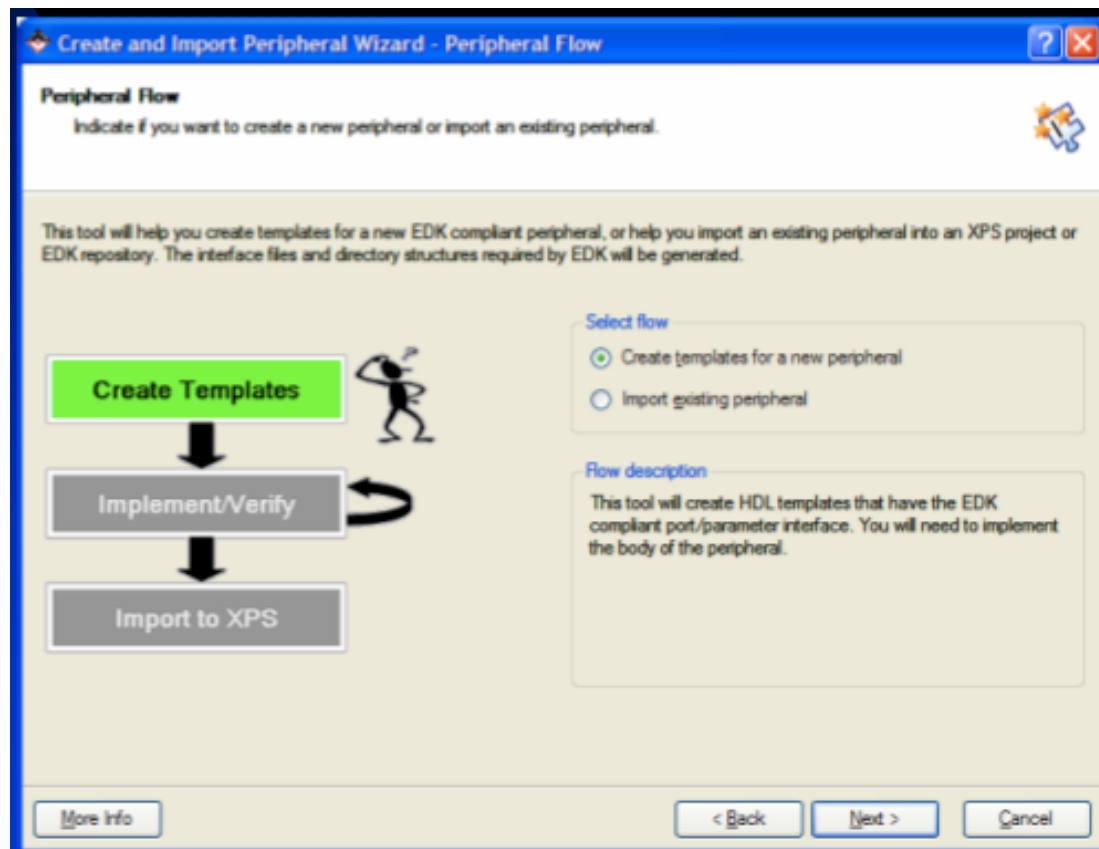
Name	Version	Description
mii_to_rmii	1.00.b	Ethernet PHY MII to Reduced MII
opb_can	1.00.a	OPB CAN Controller
opb_ethernet	1.04.a	OPB 10/100 Ethernet MAC
opb_ethernetlite	1.01.b	OPB 10/100 Ethernet MAC Lite
opb_hdlc	1.00.c	OPB HDLC Interface
opb_hdlc	2.01.a	OPB HDLC Interface

■ **Algunos periféricos son de pago, a pesar de venir incluidos en la herramienta.**

Herramienta de diseño para MicroBlaze: EDK

Descripción hardware EDK

- También es posible definir nuevos periféricos de usuario ayudados por el Peripheral Wizard.



Herramienta de diseño para MicroBlaze: EDK

Descripción hardware EDK

- El archivo MPD (Microprocessor Peripheral Description) asigna valores a los diferentes parámetros de configuración del periférico y define sus terminales y conexiones.

```
BEGIN opb_pwm

OPTION IPTYPE=PERIPHERAL

## Bus Interfaces
BUS_INTERFACE BUS = SOPB, BUS_TYPE = SLAVE

## Parameter list for the generics
PARAMETER C_BASEADDR = 0xFFFF8000, DT = std
PARAMETER C_HIGHADDR = 0xFFFF80FF, DT = std

## Ports
PORT OPB_Clk = "", DIR = in, SIGIS=CLK, BUS=SOPB
PORT OPB_Rst = OPB_Rst, DIR = in, BUS=SOPB
PORT OPB_ABus = OPB_ABus, DIR = in, VEC = [0:31], BUS=SOPB
PORT OPB_BE = OPB_BE, DIR = in, VEC = [0:3], BUS=SOPB
PORT OPB_RNW = OPB_RNW, DIR = in, BUS=SOPB
```

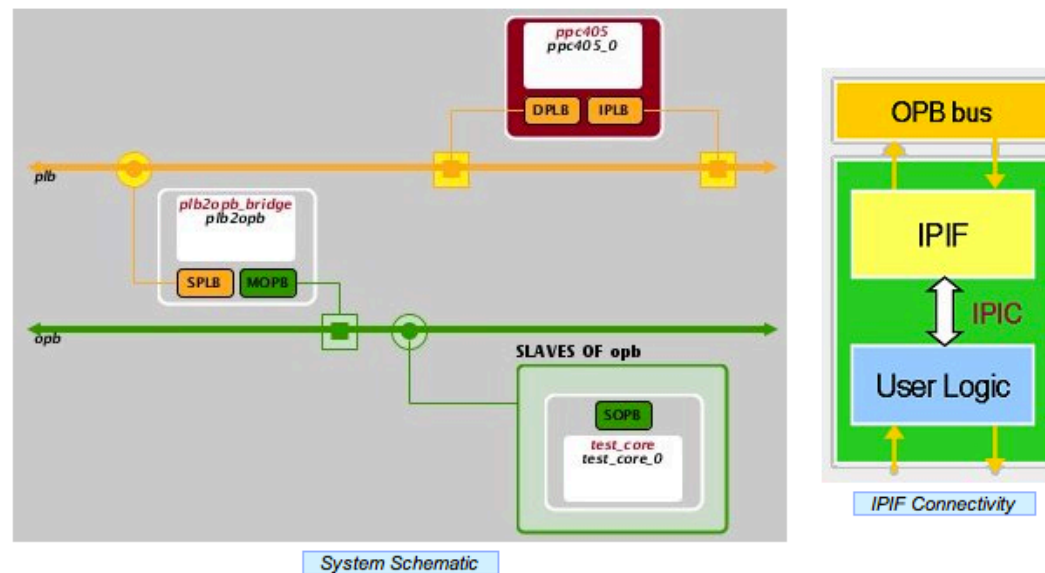
Parameters override generics in VHDL

```
entity OPB_PWM is
generic (
  C_OPB_AWIDTH   : integer           := 32;
  C_OPB_DWIDTH   : integer           := 32;
  C_BASEADDR     : std_logic_vector(0 to 31) := X"FFFA000";
  C_HIGHADDR     : std_logic_vector           := X"FFFA0FF";
  C_NO_CHANNELS  : integer range 0 to 15 := 4;
  C_MAX_RESOLUTION : integer range 4 to 32 := 16
);
```

Herramienta de diseño para MicroBlaze: EDK

Descripción hardware EDK

- ❑ Otros archivos importantes en la descripción de los periféricos son:
 - ❑ PAO (Peripheral Analysis Order): Indica a la herramienta de síntesis el orden correcto para procesar las librerías y archivos HDL.
 - ❑ BBD (Black Box Definition): Identifica los archivos que utiliza un periférico de usuario. Las listas de conexiones en formato .NGC se copian en el subdirectorio implementation del proyecto.



Herramienta de diseño para MicroBlaze: EDK

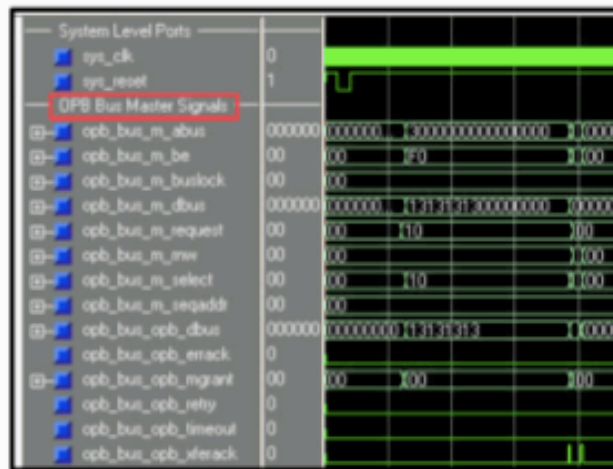
Síntesis e implementación EDK

- ❑ Mediante la opción Generate Bitstream realizamos la síntesis y la implementación del hardware.
 - ❑ Al final del proceso en el subdirectorio implementation tenemos los archivos:
 - ❑ system_map.mrp: Resumen de la asignación de recursos de la FPGA.
 - ❑ system.bit: Archivo de programación de la FPGA.
 - ❑ system.ucf: Archivo con la definición de los terminales.
-

Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

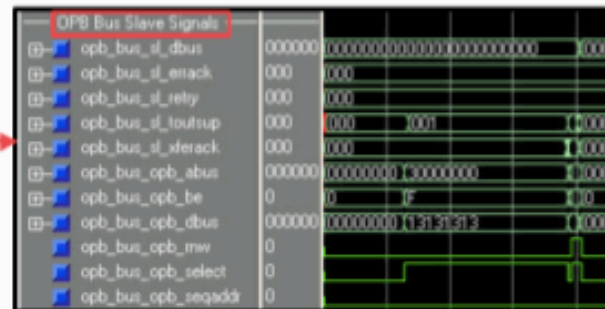
- Con ModelSim (PE o SE) podemos simular el hardware del sistema empotrado. Con ModelSim XE sólo podremos simular partes del mismo.



Waveform showing the **OPB** signals generated by the **Master BFM**. For this simulation, the device under test (**my_opb_peripheral**) is placed at address 0x30000000 in the memory.

The simulation consists of a write to the device under test followed by a read.

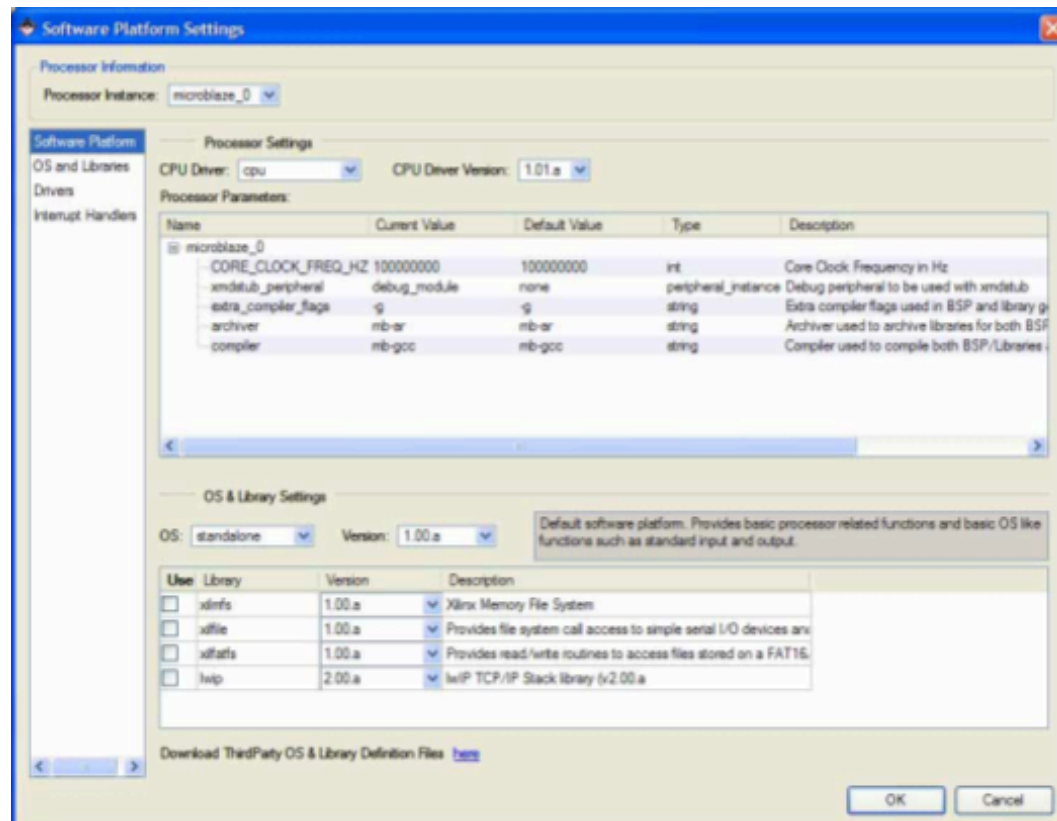
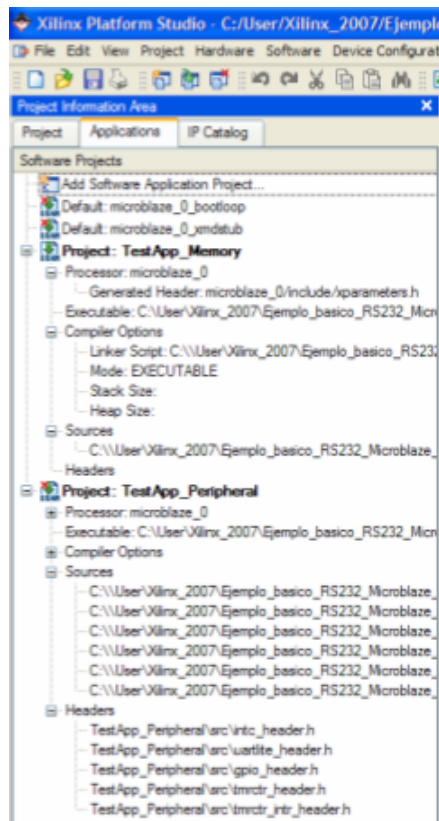
Waveform showing the **OPB** signals generated by the device under test.



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

- Mediante XPS podemos definir las opciones del software y podremos añadir aplicaciones software en nuestro sistema empujado.



Herramienta de diseño para MicroBlaze: EDK

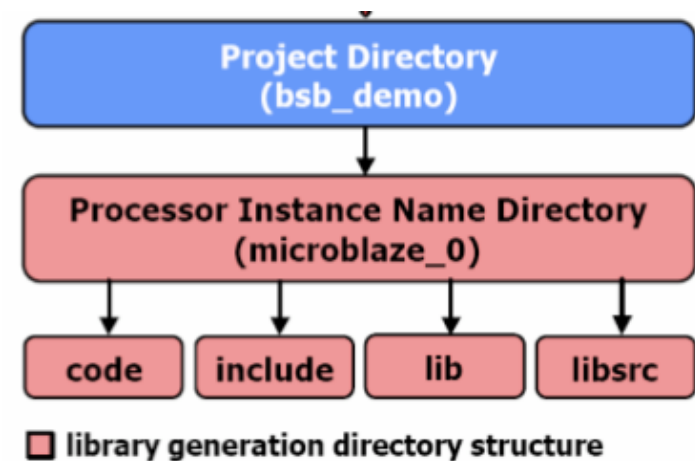
Síntesis e implementación EDK

- ❑ Necesitamos definir las bibliotecas y el sistema operativo para nuestra aplicación.
 - ❑ Xilinx proporciona tres bibliotecas:
 - ❑ Matemática (libm)
 - ❑ Estándar del lenguaje C (libc)
 - ❑ Las funciones de esta biblioteca están disponibles automáticamente.
 - ❑ Específica de Xilinx para C (libxil)
 - ❑ Funciones para manejo de archivos: LibXil File.
 - ❑ Funciones para manejo de memoria: LibXil Mfs.
 - ❑ Funciones para manejo de redes: LibXil Net.
 - ❑ Xilinx proporciona el sistema operativo Xilinx Micro Kernel.
-

Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

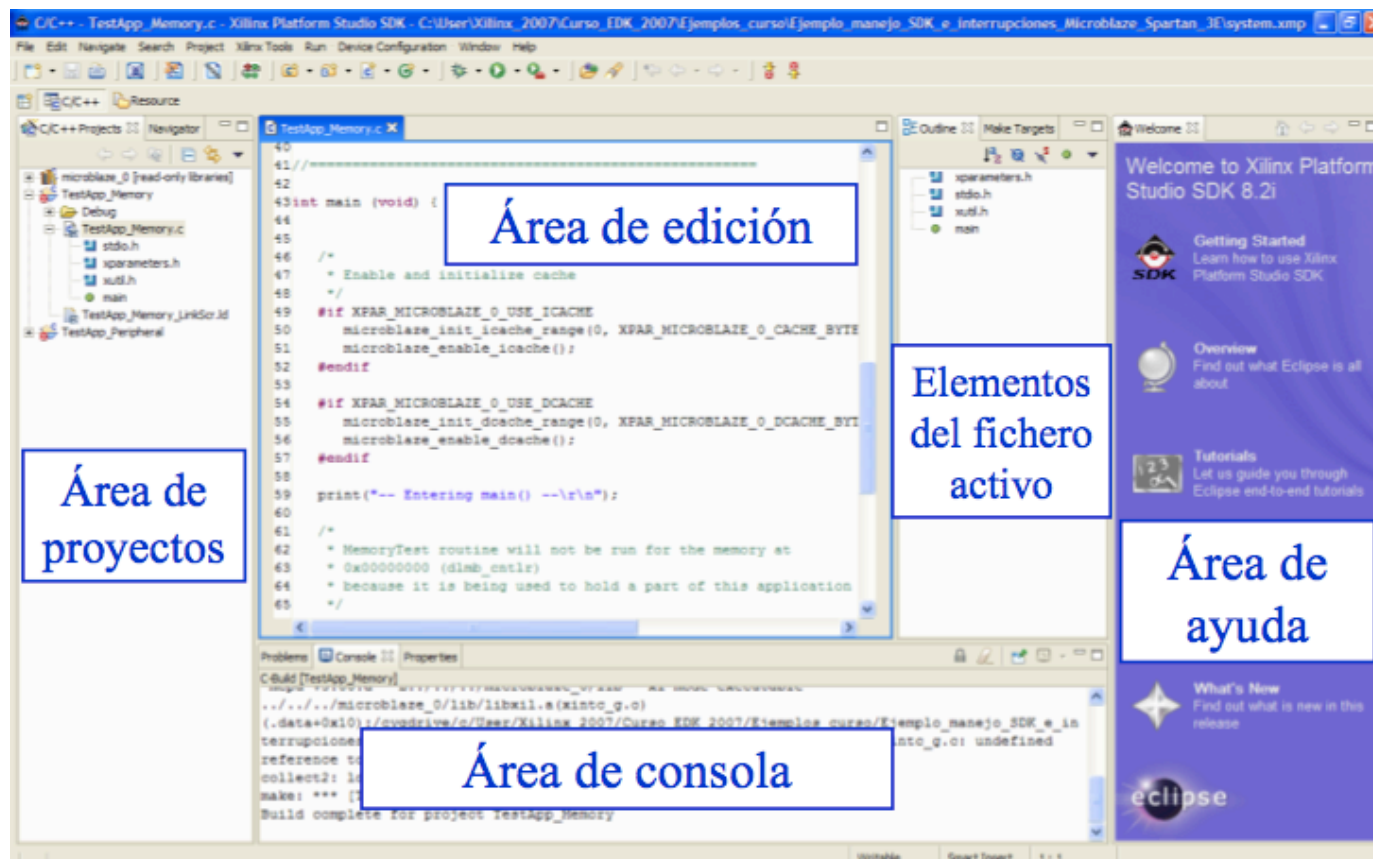
- ❑ Mediante la opción Generate Libraries and BSPs de la herramienta XPS generamos los archivos necesarios para trabajar con las bibliotecas seleccionadas.
- ❑ Estructura de directorios creada por XPS.
 - ❑ code: Archivos de salida del compilador.
 - ❑ include: Archivos de cabecera (*.h)
 - ❑ lib: Bibliotecas C estándar.
 - ❑ libsrc: Driver software de cada periférico.
- ❑ El archivo xparameters.h contiene las direcciones base de cada periférico y sus identificadores.



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

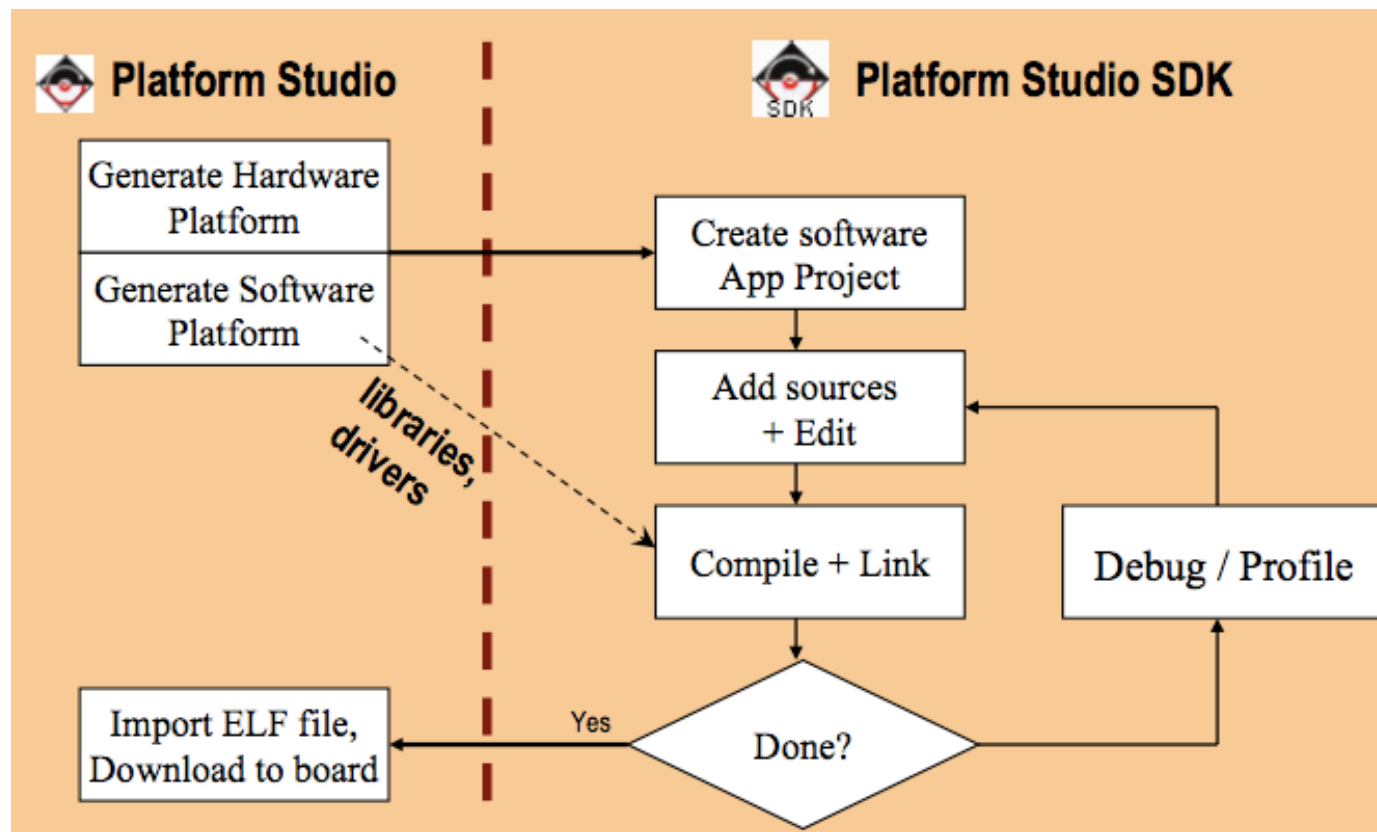
- El entorno EDK dispone de la herramienta SDK (Software Development Kit)



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

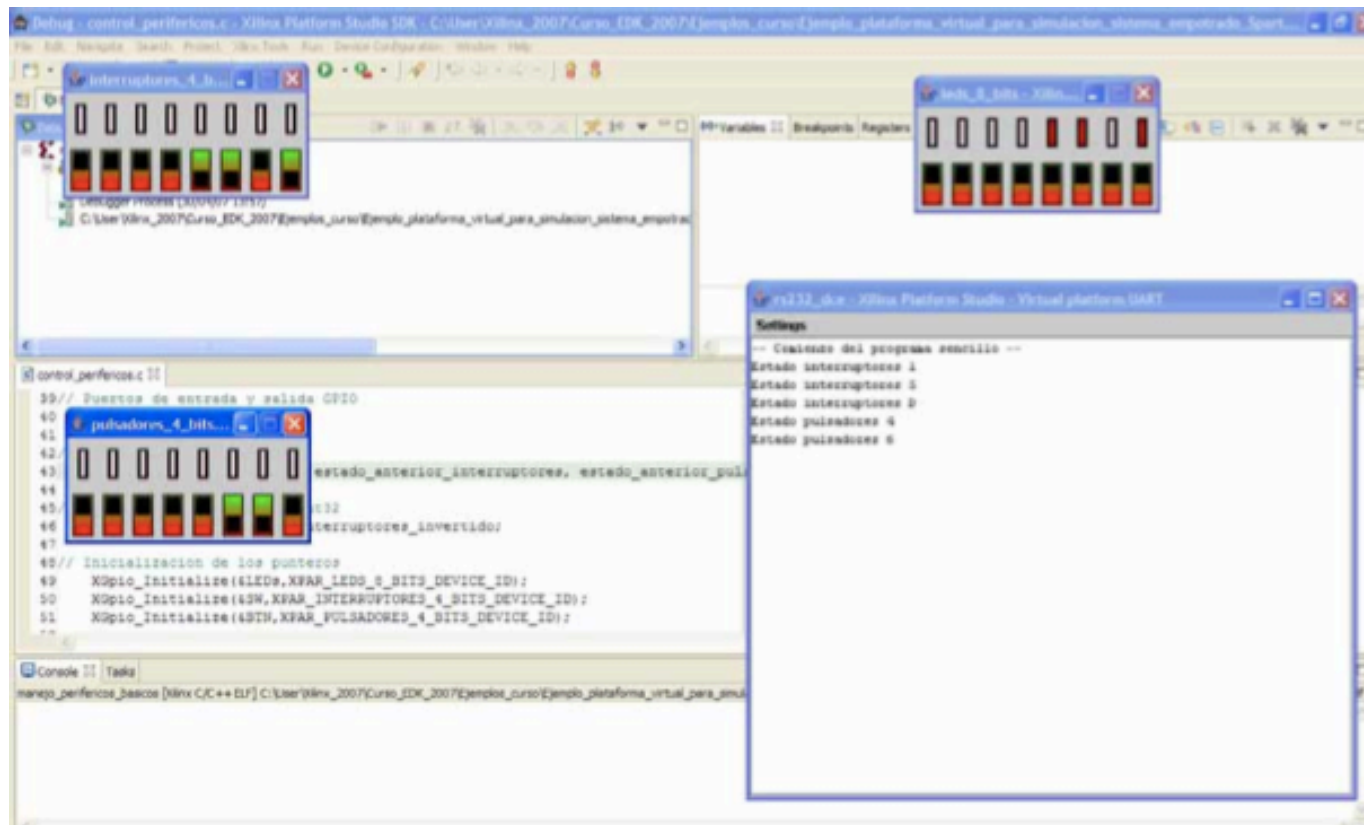
- Etapas de diseño de software con SDK.



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

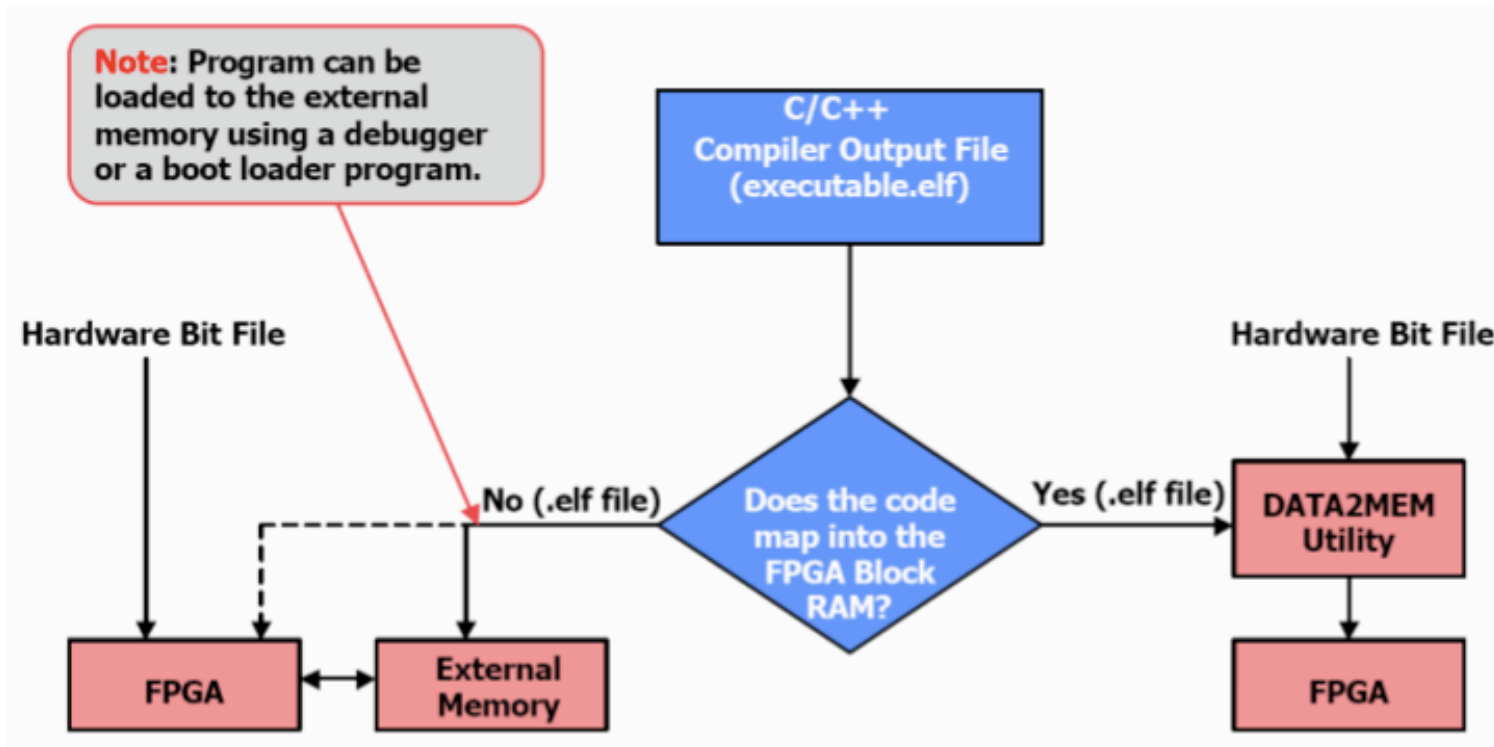
- Podemos simular el software con la herramienta de depuración que incluye EDK.



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

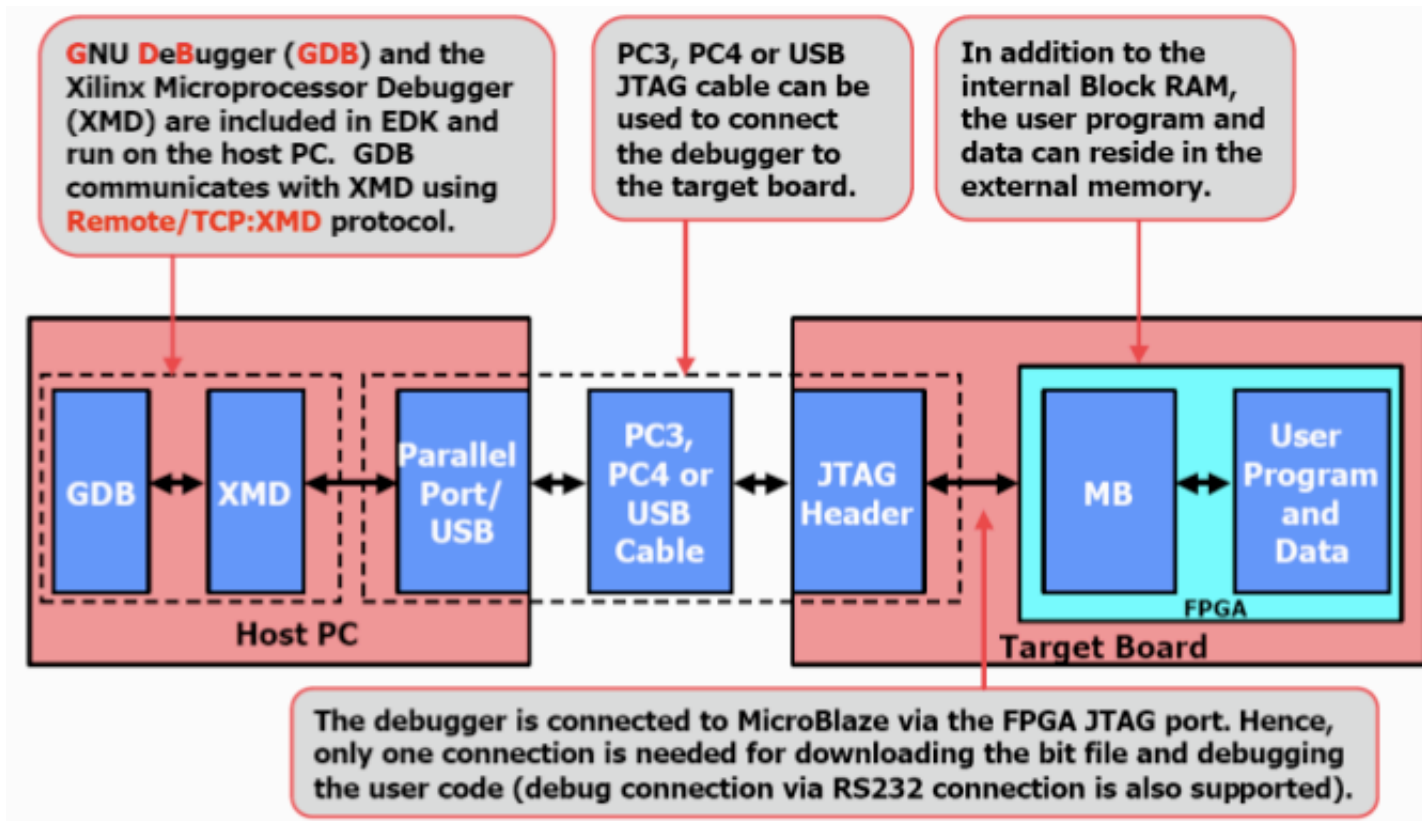
- Una vez desarrollados el hardware y el software del sistema empotrado debemos combinarlos para disponer de la aplicación completa y programar la FPGA.



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

- Una vez empotrado el sistema en la FPGA podemos depurar el hardware y el software con GDB (GNU DeBugger).



Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

- ❑ Ventana de depuración del software en SDK.

The screenshot displays the Xilinx Platform Studio SDK debugger interface. The main window is titled 'Debug: temporizador.c - Xilinx Platform Studio SDK'. The interface is divided into several panes:

- Área de depuración:** The top-left pane shows the 'Debug' tree with a tree view of the running process and threads.
- Área de información:** The top-right pane shows 'M* Variables', 'Breakpoints', 'Registers', 'XMD Console', and 'Memory'. The 'Variables' pane shows 'count_mod_3 = 6060'.
- Área de código fuente:** The bottom-left pane shows the source code for 'temporizador.c' with line numbers 84 to 96. Line 88 is highlighted: 'XGpio_Initialize(&gpio, XPAR_LEDS_8BIT_DEVICE_ID);'.
- Área de desensamblado:** The bottom-right pane shows the disassembled code for the highlighted line, including instructions like 'addik r5, r0, 5088' and 'brlid r15, 1392'.
- Área de consola:** The bottom-most pane shows the console output: 'whatis count_mod_3' followed by 'type = int'.

Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

- ❑ Ventana de depuración del hardware mediante ChipScope.

The screenshot displays the ChipScope Pro Analyzer interface. The main window is titled "ChipScope Pro Analyzer [configuracion_Chipscope]". The interface is divided into several panels:

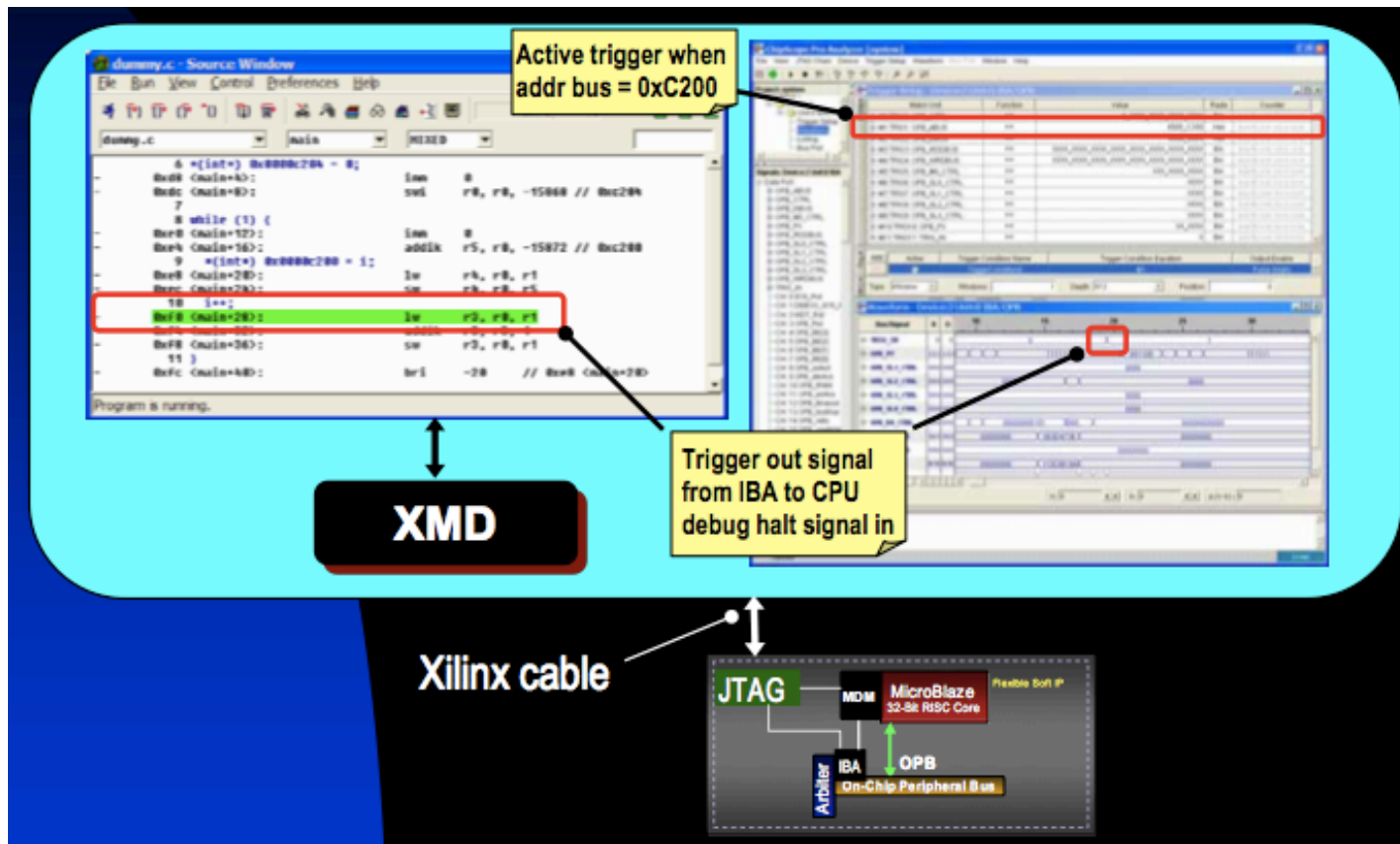
- Project:** configuracion_Chipscope
- JTAG Chain:** Shows a tree view of the hardware configuration, including DEV0 MyDevice0 (XC3S500E) and its sub-units.
- Signals:** Lists various signals available for capture, such as OPB_DBUS, OPB_ABUS, and OPB_CTRL.
- Trigger Setup:** A table defining trigger conditions for the hardware. The table is as follows:

Match Unit	Function	Value	Radix	Counter
M0 TRIG0: OPB_CTRL	==	X_XXXX_XXXX_XXXX_XXXX	Bin	disabled
M1 TRIG1: OPB_ABUS	==	4000_0000	Hex	disabled
M2 TRIG2: OPB_DBUS	==	0000_0005	Hex	disabled
- Waveform:** Displays a digital waveform for the selected signals. The x-axis represents time in samples, with markers at -16, -11, -6, -1, 4, 9, and 14. The y-axis lists signals: OPB_DBUS, OPB_ABUS, and OPB_RST. The OPB_ABUS signal shows a transition from 00000000 to 40000000 at time -1.
- Info:** A status bar at the bottom indicates "INFO - Device 0 Unit 0: Waiting for core to be armed".

Herramienta de diseño para MicroBlaze: EDK

Síntesis e implementación EDK

- Codepuración hardware-software mediante SDK y ChipScope.



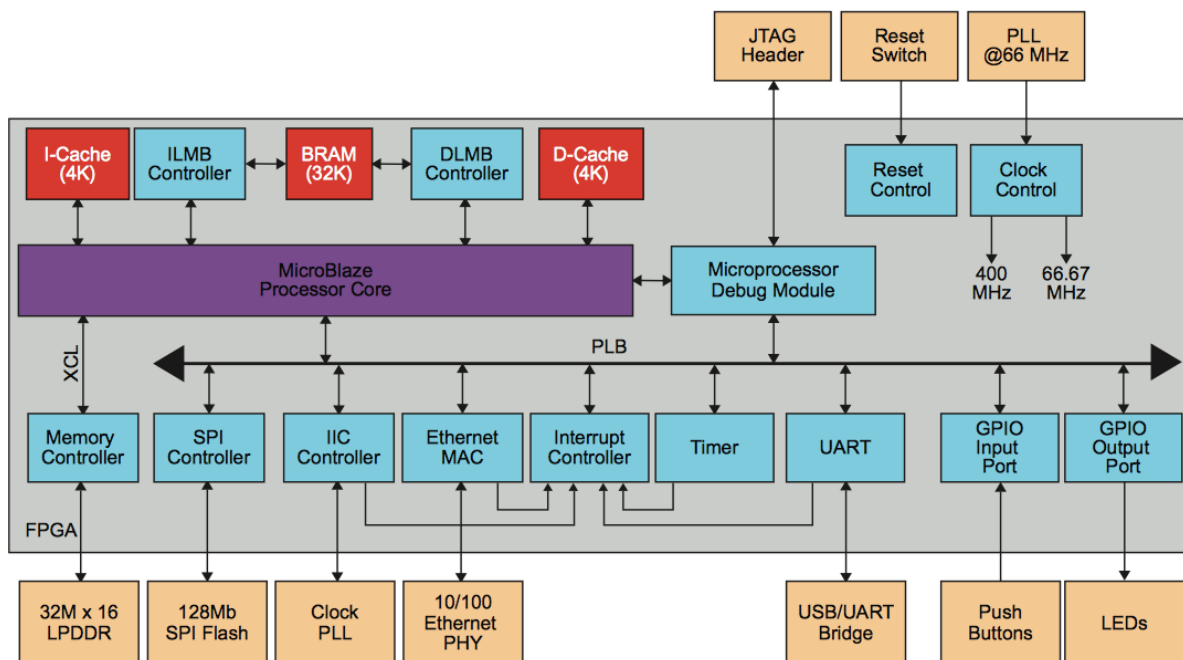
Creación de aplicaciones software

Una de las complejidades de empotrar MicroBlaze en una FPGA reside en conocer los periféricos que usamos para realizar la programación y acceso a los mismos.

Vamos en éste segundo ejercicio a crear una aplicación básica que nos permita acceder a cualquier periférico que queramos controlar con MicroBlaze. Usaremos XPS para generar la aplicación software, creando el archivo de programación que verificaremos de manera hardware. Así mismo, usaremos un linker script simple que nos permita automatizar todo el proceso.

El primer paso será crear un directorio, donde guardemos nuestros proyectos, con el nombre de éste ejercicio. Copiaremos todos los archivos del anterior ejercicio a éste directorio. Así no tendremos que generar de nuevo todo el hardware con el que vamos a trabajar, sólo crearemos el software.

Una vez realizado, abriremos XPS y el proyecto copiado. Bien desde EDK o desde el Explorador de Windows haciendo doble clic en system.xmp

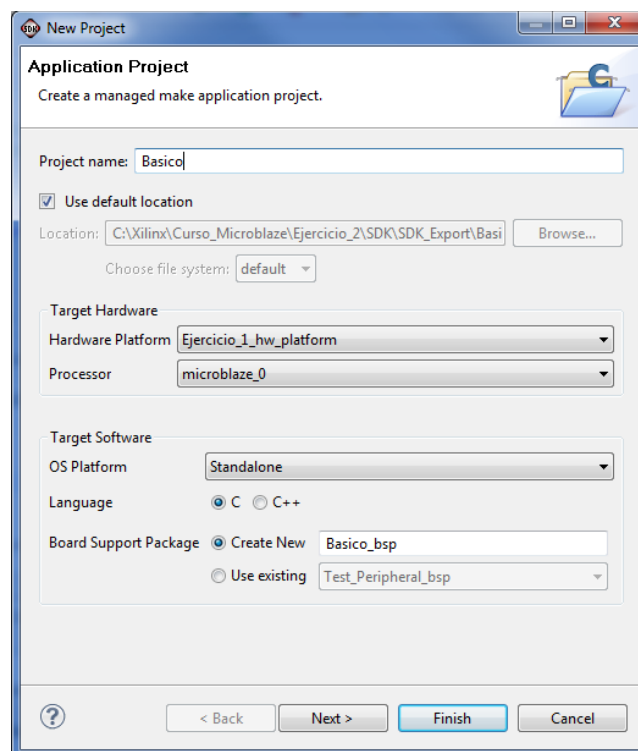


Creación aplicación software - SDK

Lanzamos desde EDK, al igual que hicimos en el Ejercicio 1, el hardware para trabajar en SDK. Un vez en SDK usamos la opción File->New->Application Project y le asignaremos un nombre a nuestro proyecto software (Basico).

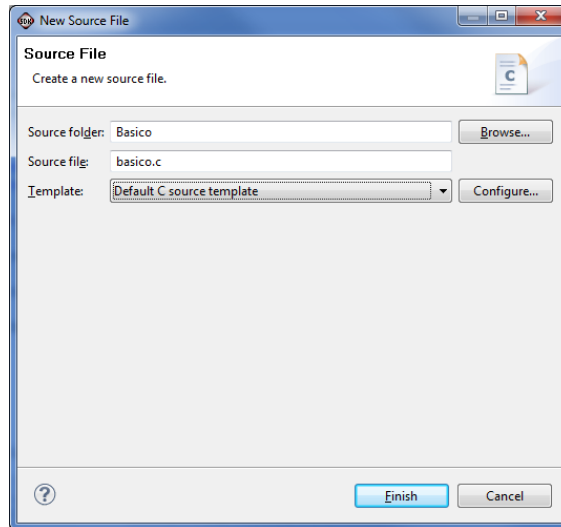
Teniendo cuidado con tener elegido el procesador correcto (microblaze_0) y seleccionando el hardware generado en el ejercicio anterior con EDK (Ejercicio_1_hw_platform).

En cuanto al soporte de la placa que estamos empleando (Board Support Package) podríamos emplear el generado anteriormente (Ejercicio_1_bsp_0) ya que no ha sido modificado, pero para conocer mejor la herramienta se va a generar uno nuevo con los archivos de soporte (Basico_bsp) de la tarjeta LX9 que se está empleando.



Una vez realizados estos pasos, le damos a Next y le decimos que nos cree una Aplicación Software Vacía puesto que vamos a ser nosotros quienes creamos el Programa Software (Empty Application).

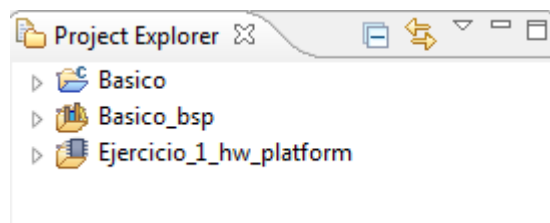
Con la opción File -> New->Source File, crearemos un archivo C donde podremos introducir nuestro código, para ello, debemos indicar el nombre del proyecto en el que deseamos incluir el archivo, el nombre del mismo (basico.c) y el lenguaje que se va a emplear para su programación (C).



El primer paso, es añadir los archivos de librerías con los cuales vamos a trabajar. Todos éstos archivos los encontramos en la carpeta del proyecto Board Package Support (Basico_bsp \microblaze_0\include).

Por tanto, se pueden eliminar de la ventana de Project Explorer los proyectos que no vamos a emplear en este ejercicio conservando el proyecto donde vamos a realizar la programación (Basico), el de soporte de la tarjeta (Basico_bsp) y el hardware que generamos en el Ejercicio 1 y que vamos a emplear en este caso (Ejercicio_1_hw_platform).

Pinchar sobre cada uno de ellos y con el botón derecho usar la opción Delete. Tiene que quedarnos algo como lo de esta imagen.

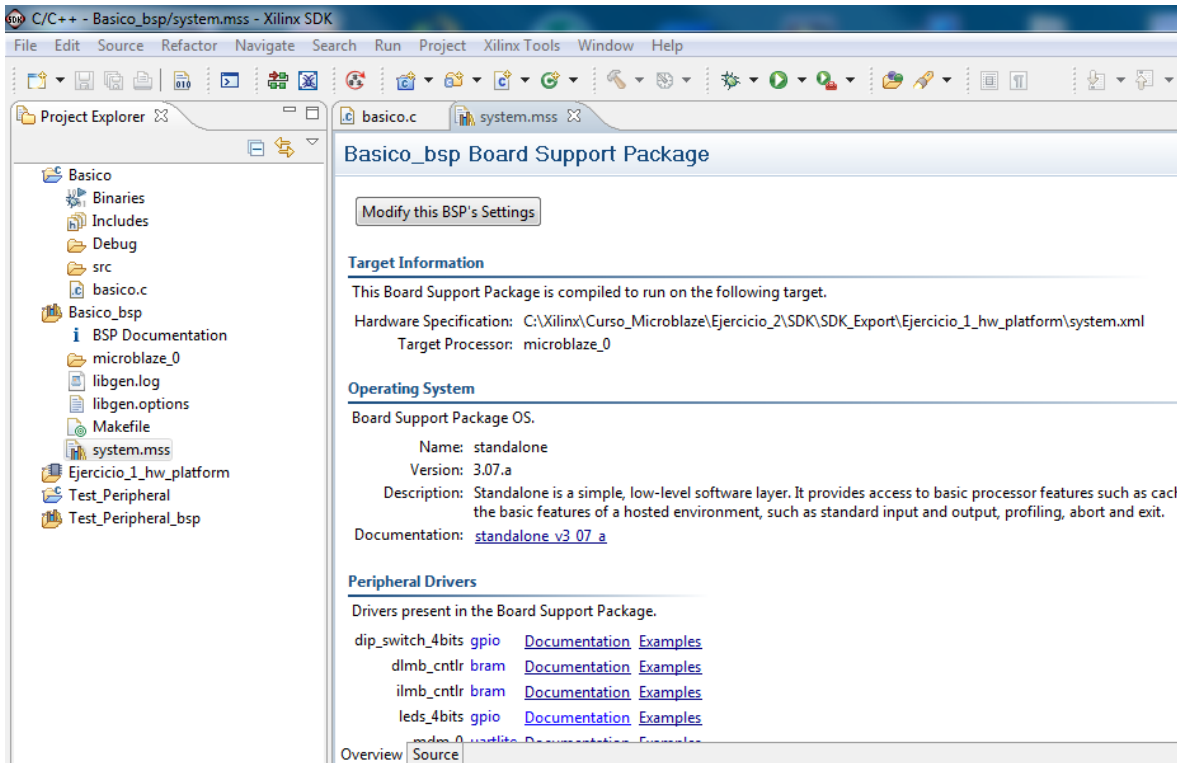


Para éste ejercicio, incluiremos las librerías xparameters y xgpio.

```
1 #include "xparameters.h"
2 #include "xgpio.h"
```

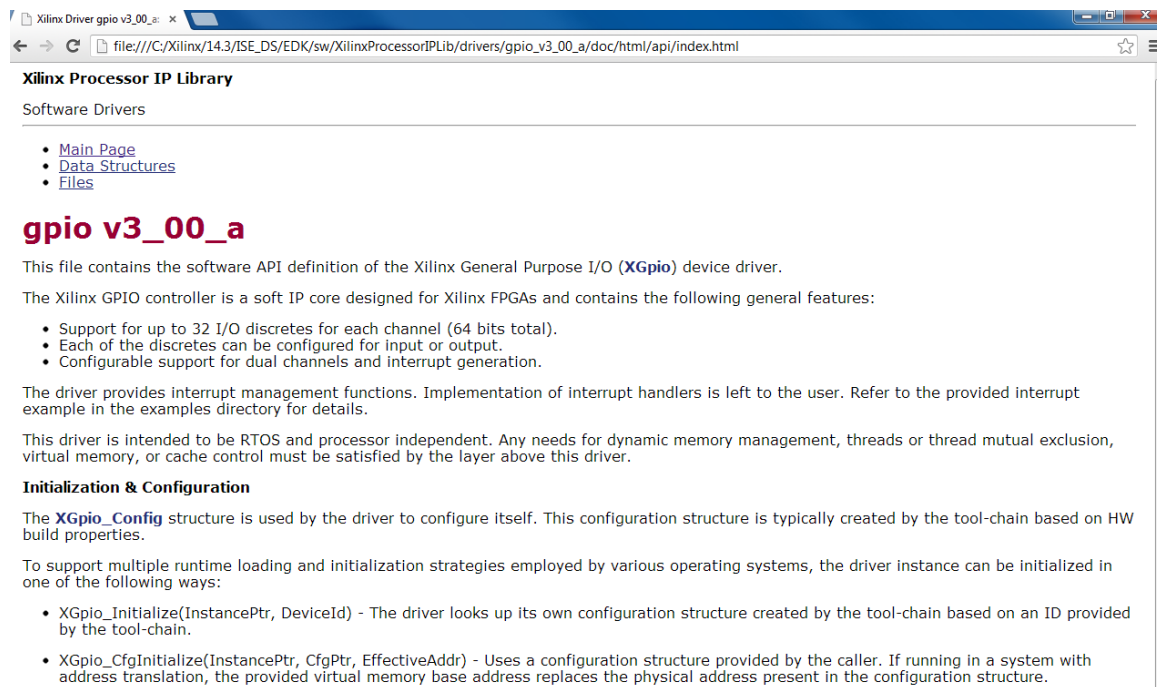
El siguiente paso es conocer como se puede inicializar un periférico, si no lo hacemos, el microprocesador no sabrá sobre que periférico se está actuando. Si queremos usar los LEDs, podemos navegar en la pestaña Project Explorer, dentro de Basico_bsp clicando sobre el archivo system.mss, en el apartado Peripheral Drivers seleccionar leds_4bits gpio Documentation.

A continuación, se abrirá en el navegador una página con la descripción de la API del periférico GPIO.

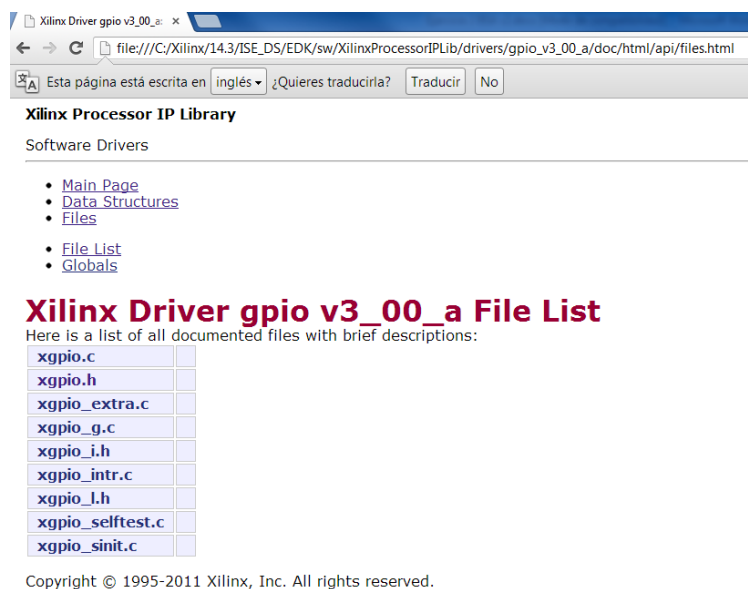


En ésta documentación encontramos todos los recursos para poder actuar sobre el soft IP core diseñado por Xilinx que nos permite acceder al periférico.

En la ventana que se ha abierto en el navegador seleccionamos Files.



A continuación, seleccionamos el archivo xgpio.h

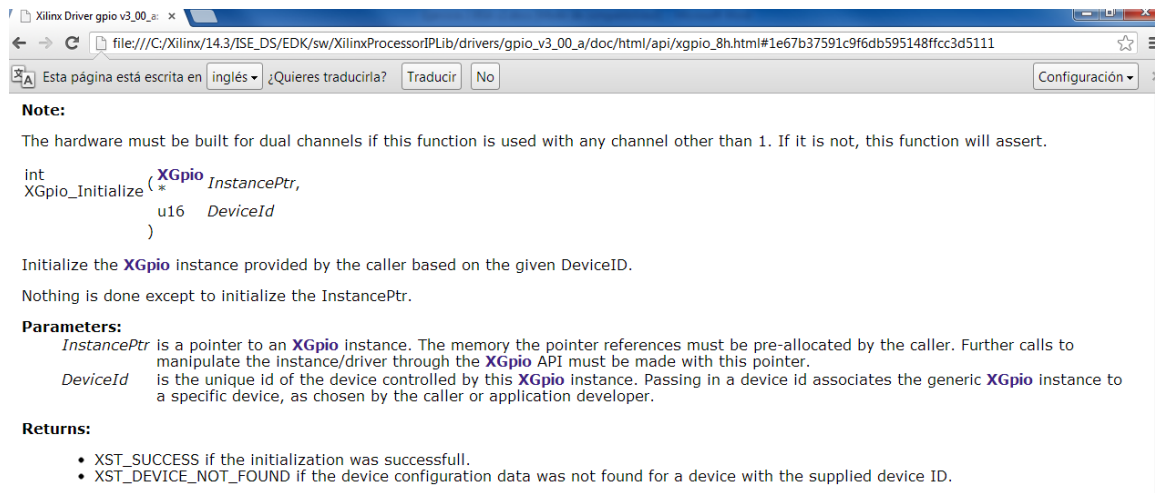


Si vamos al apartado Functions podremos ver la lista de funciones que podemos usar.

Functions

int	XGpio_Initialize	(XGpio *InstancePtr, u16 DeviceId)
XGpio_Config *	XGpio_LookupConfig	(u16 DeviceId)
int	XGpio_CfgInitialize	(XGpio *InstancePtr, XGpio_Config *Config, u32 EffectiveAddr)
void	XGpio_SetDataDirection	(XGpio *InstancePtr, unsigned Channel, u32 DirectionMask)
u32	XGpio_GetDataDirection	(XGpio *InstancePtr, unsigned Channel)
u32	XGpio_DiscreteRead	(XGpio *InstancePtr, unsigned Channel)
void	XGpio_DiscreteWrite	(XGpio *InstancePtr, unsigned Channel, u32 Mask)
void	XGpio_DiscreteSet	(XGpio *InstancePtr, unsigned Channel, u32 Mask)
void	XGpio_DiscreteClear	(XGpio *InstancePtr, unsigned Channel, u32 Mask)
int	XGpio_SelfTest	(XGpio *InstancePtr)
void	XGpio_InterruptGlobalEnable	(XGpio *InstancePtr)
void	XGpio_InterruptGlobalDisable	(XGpio *InstancePtr)
void	XGpio_InterruptEnable	(XGpio *InstancePtr, u32 Mask)
void	XGpio_InterruptDisable	(XGpio *InstancePtr, u32 Mask)
void	XGpio_InterruptClear	(XGpio *InstancePtr, u32 Mask)
u32	XGpio_InterruptGetEnabled	(XGpio *InstancePtr)
u32	XGpio_InterruptGetStatus	(XGpio *InstancePtr)

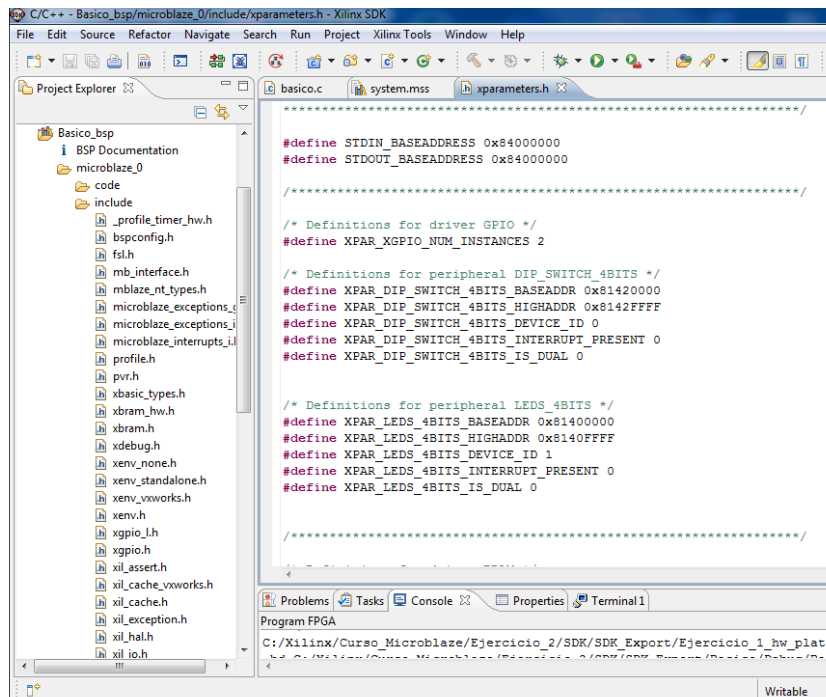
Como lo primero que queremos es inicializar nuestra conexión con el periférico, debemos usar las funciones de XGpio_Initialize().



Contiene una detallada descripción de la función XGpio_Initialize(). Hay 2 parámetros que debemos configurar:

- InstancePtr: pondremos (&gp_out) que define una variable de tipo puntero que nos permite acceder a éste periférico.
- DeviceId: pondremos (XPAR_LEDS_4BITS_DEVICE_ID) que es la identificación de nuestro dispositivo.

Para encontrar esta identificación, es necesario acudir a xparameters.h. Podemos abrirla en la ruta Basico_bsp -> microblaze_0 -> include en la pestaña Project Explorer



Por tanto, nuestro código debería tener este aspecto en este punto:

```
#include "xparameters.h"
#include "xgpio.h"

int main(void)
{
    XGpio gp_out;

    XGpio_Initialize(&gp_out, XPAR_LEDS_4BITS_DEVICE_ID);
```

Para definir todos los bits del bus de conexión con los LEDs como salidas, deberemos añadir la siguiente línea:

```
XGpio_SetDataDirection (&gp_out, 1, 0x0)
```

Buscar la documentación y entender la línea.

Una vez realizado lo anterior, ya podemos implementar cualquier función en C estándar. En éste caso, vamos a crear un simple contador que nos permita recorrer los LEDs encendiéndolos de uno en uno.

```
#include "xparameters.h"
#include "xgpio.h"

int main(void)
{
    XGpio gp_out;

    XGpio_Initialize(&gp_out, XPAR_LEDS_4BITS_DEVICE_ID);

    XGpio_SetDataDirection (&gp_out, 1, 0x00);

    int i=0;
    int j=0;

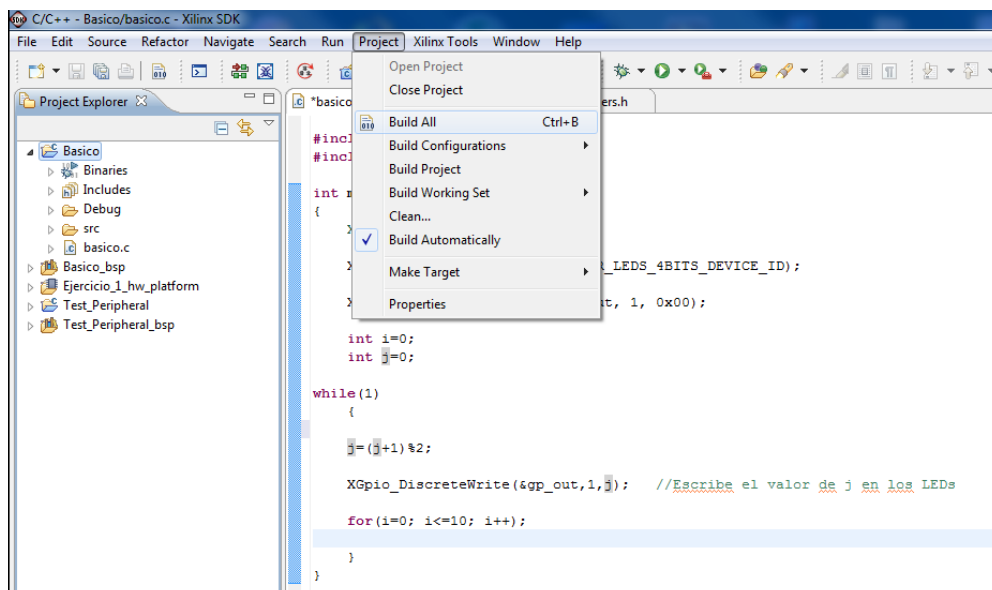
    while(1)
    {
        j=(j+1)%2;

        XGpio_DiscreteWrite(&gp_out,1,j); //Escribe el valor de j en los LEDs

        for(i=0; i<=10; i++);
    }
}
```

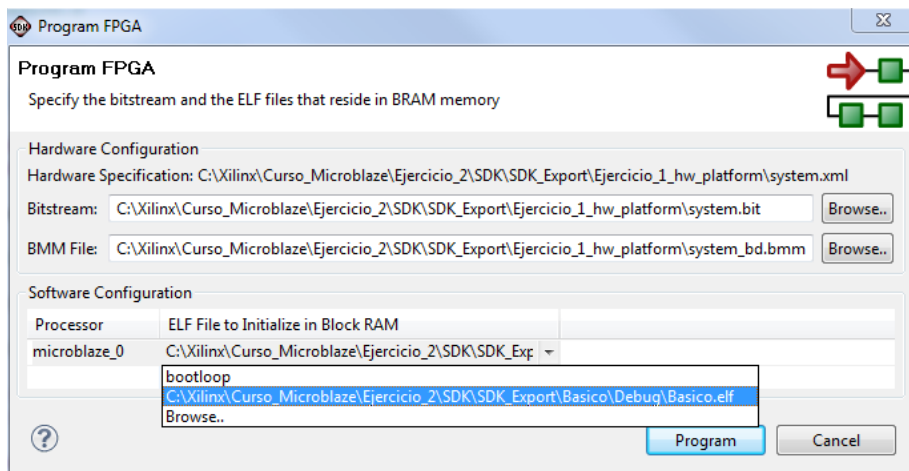
Fijaros en la nueva función usada `XGpio_DiscreteWrite()`, comprender lo que hace.

Ya estamos en disposición de compilar nuestro primer programa. Usaremos la opción Software -> Build All.



Podemos observar en la ventana de la consola el tamaño que ocupa nuestro ejecutable.

Ahora ya tenemos lo necesario para programar la FPGA y ver como funciona nuestro código. Para ello debemos seleccionar el Bitstream y el BMM File generados en el hardware del Ejercicio 1 que hemos copiado a nuestro proyecto Ejercicio_2 y el software generado para este segundo ejercicio (Basico.elf).



¿Es correcto el funcionamiento? ¿La temporización es la adecuada? Si no es así, modificar el código para que veamos como se recorren los LEDs.

Crear un nuevo Proyecto en C basado en la plataforma hardware y software ya creadas para mostrar en los LEDs el estado de los interruptores. ¿Qué tendrías que modificar del siguiente código para que funcione?

```
#include "xparameters.h"
#include "xgpio.h"
#include "xutil.h"

int main(void)
{
    XGpio interruptor;

    int i, estado_interruptor;

    //define los LEDs

    xil_printf("-- Inicio de programa --\r\n");

    XGpio_Initialize(&interruptor, XPAR_DIP_SWITCH_4BITS_DEVICE_ID);
    XGpio_SetDataDirection(&interruptor, 1, 0xff);

    // inicializa y dale la dirección a los LEDs

    while(1)
    {
        // lee el estado de los interruptores

        // delay para controlar el tiempo de envío por terminal
        for (i=0; i<1000000; i++)
        // muestra el estado de los pulsadores en los LEDs
        XGpio_DiscreteWrite(&LEDs4_Bit, 1, estado_interruptor);
    }
}
```

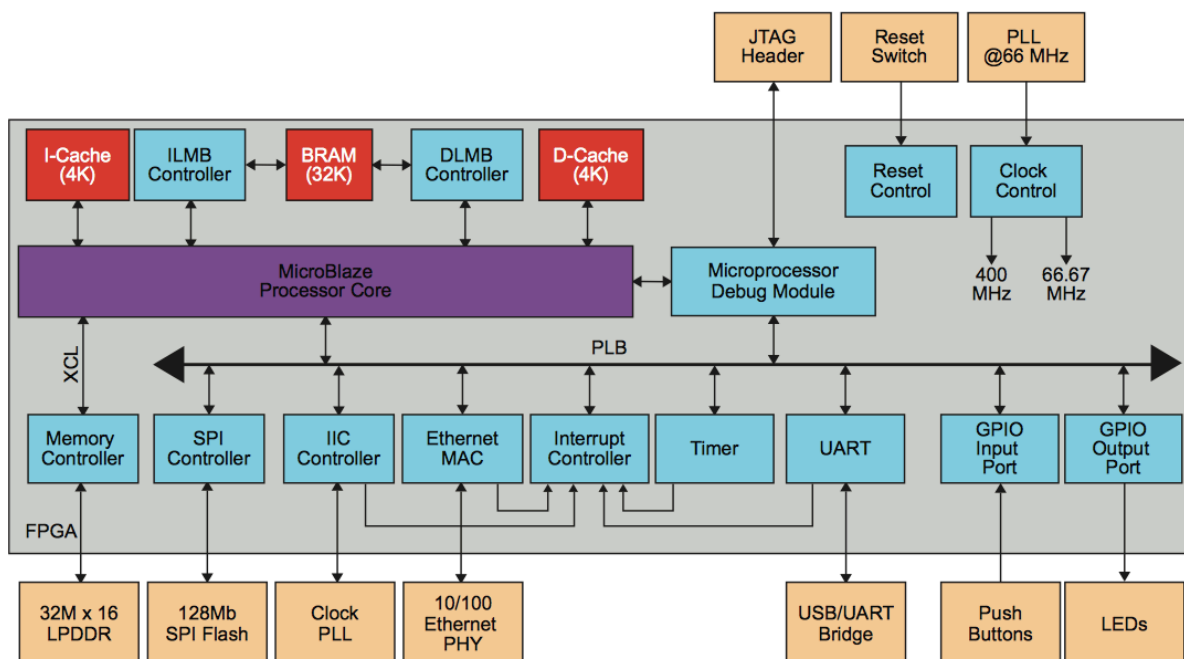
Configuración periféricos GPIO

En este cuarto diseño con MicroBlaze vamos a crear un sistema empotrado hardware configurando periféricos GPIO (General Purpose Input Output).

El desarrollo de este ejercicio nos llevará a crear un diseño hardware con MicroBlaze utilizando de nuevo BSB pero añadiendo y configurando los periféricos GPIO. Crearemos 2 GPIO para la lectura de los interruptores y un pulsador externo y 1 GPIO más para la escritura en los LEDs.

Todo ello lo mostraremos en Terminal por el Puerto Serie de la placa de desarrollo LX9. Así mismo, crearemos una aplicación software para gestionar estos periféricos.

El diagrama de bloques que tendrá esta aplicación es el siguiente.

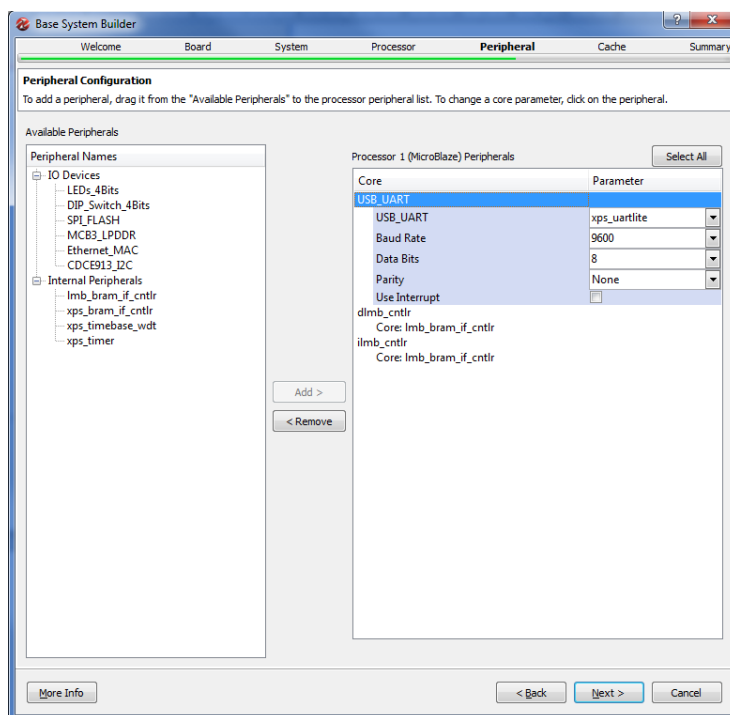


Creación sistema empotrado

Siguiendo los pasos del Ejercicio_1, creamos un nuevo proyecto con el nombre Ejercicio_4. Definiendo Microblaze como nuestro microprocesador empotrado, la misma placa de desarrollo usada anteriormente, la misma frecuencia y el mismo tamaño de memoria BRAM.

En este caso también usaremos el puerto USB_UART, mientras que desactivaremos la selección de los periféricos (LEDs y Switches). Igualmente, desactivaremos el uso de la SPI Flash, de la DDR SDRAM y de Ethernet.

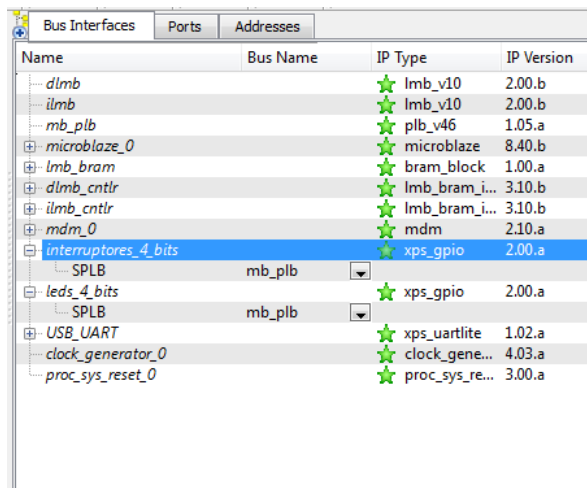
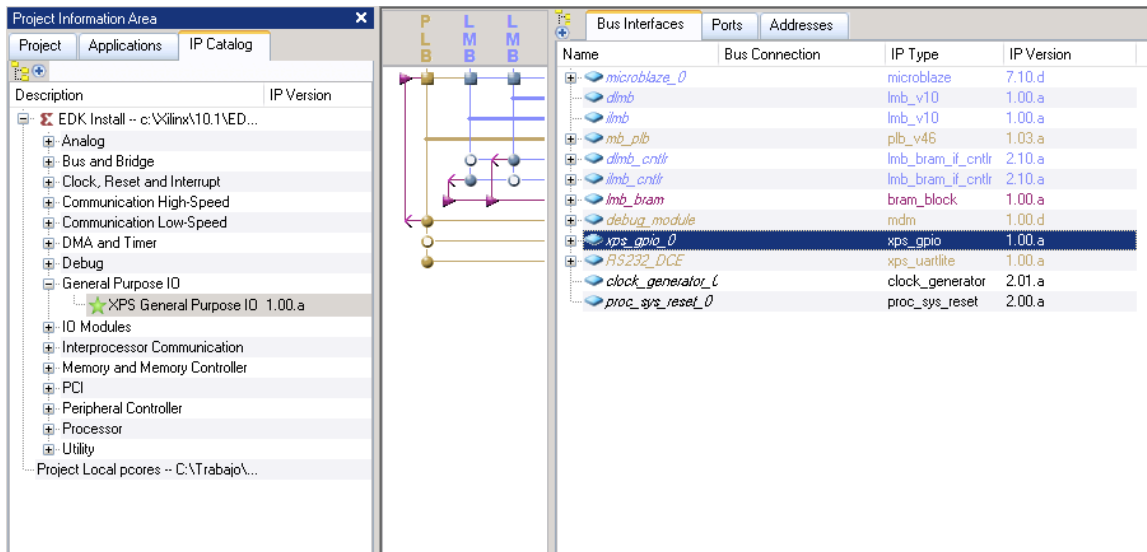
Debemos obtener lo siguiente y finalizamos la generación de nuestro sistema empotrado.



Una vez abierto el entorno XPS con el nuevo proyecto creado, implementar el hardware con la opción Hardware -> Generate Bitstream para comprobar que no tenemos ningún error.

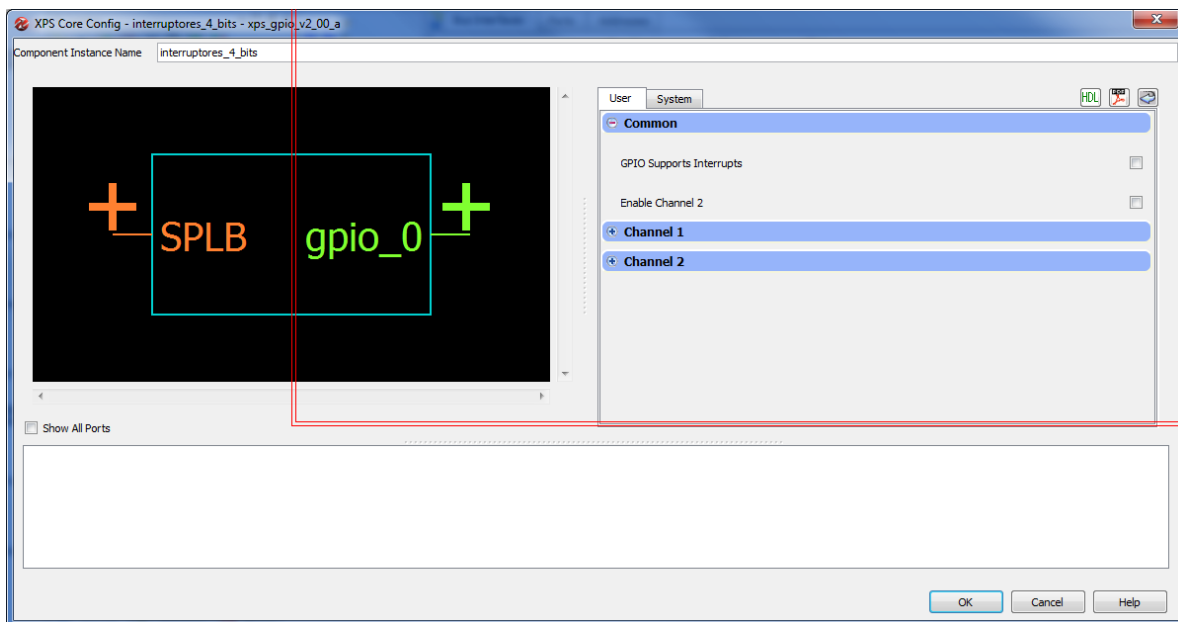
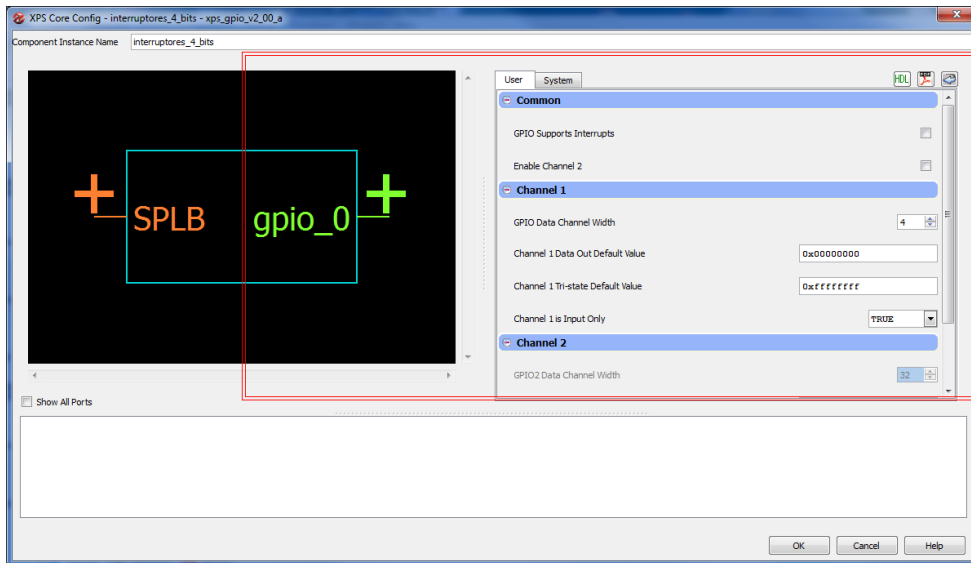
Ahora es el momento de añadir un nuevo periférico a nuestro sistema empotrado. Vamos a conectar los interruptores de 4 bits de la placa LX9. Se tratan como GPIO (XPS General Purpose IO).

Para ello, hacemos doble click sobre su nombre en la pestaña IP Catalog y lo añadimos a nuestro sistema empotrado. Le llamaremos interruptores_4_bits.



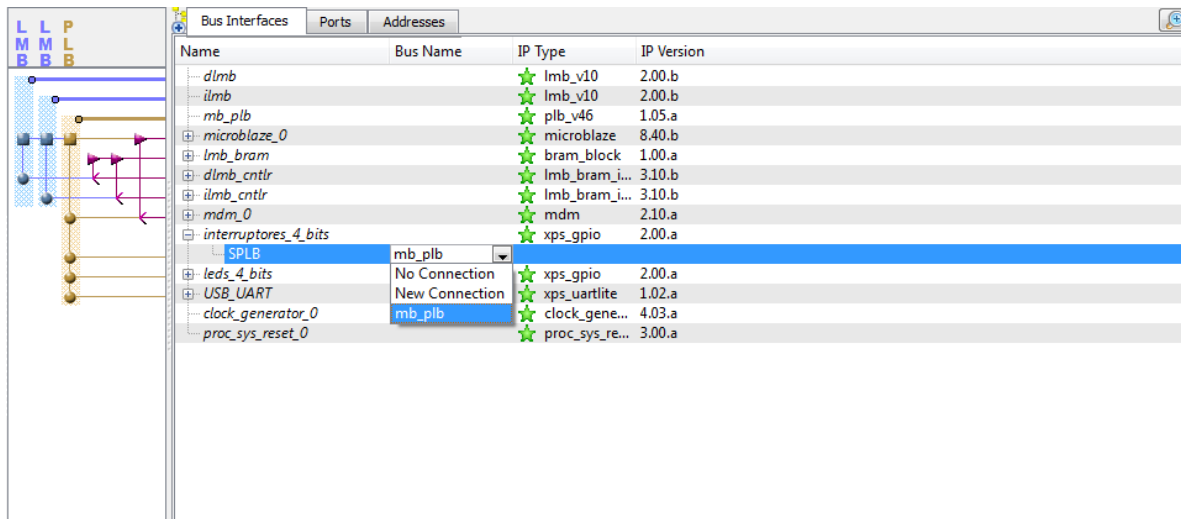
Ahora vamos a configurar los parámetros del periférico creado. Para ello, pinchamos dos veces sobre su nombre y abrimos el cuadro de diálogo correspondiente.

En la pestaña USER desplegamos las opciones del Channel 1 y seleccionamos un tamaño de bus de 4 bits y lo configuramos como solo entrada.

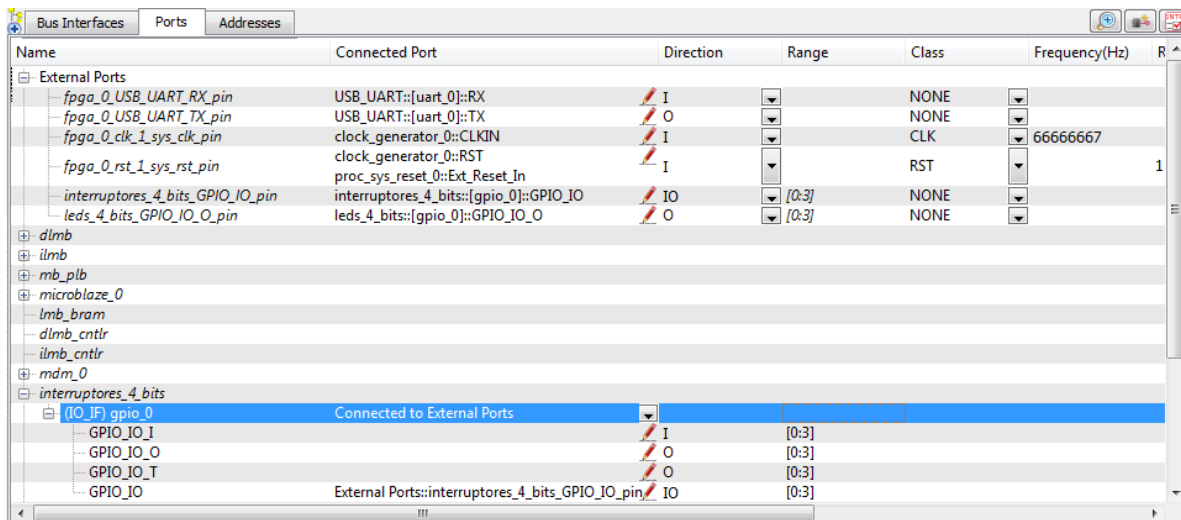


De la pestaña Channel 2 podemos dejar los valores por defecto puesto que dicho canal no se utiliza.

Ahora conectamos el periférico creado al bus PLB.



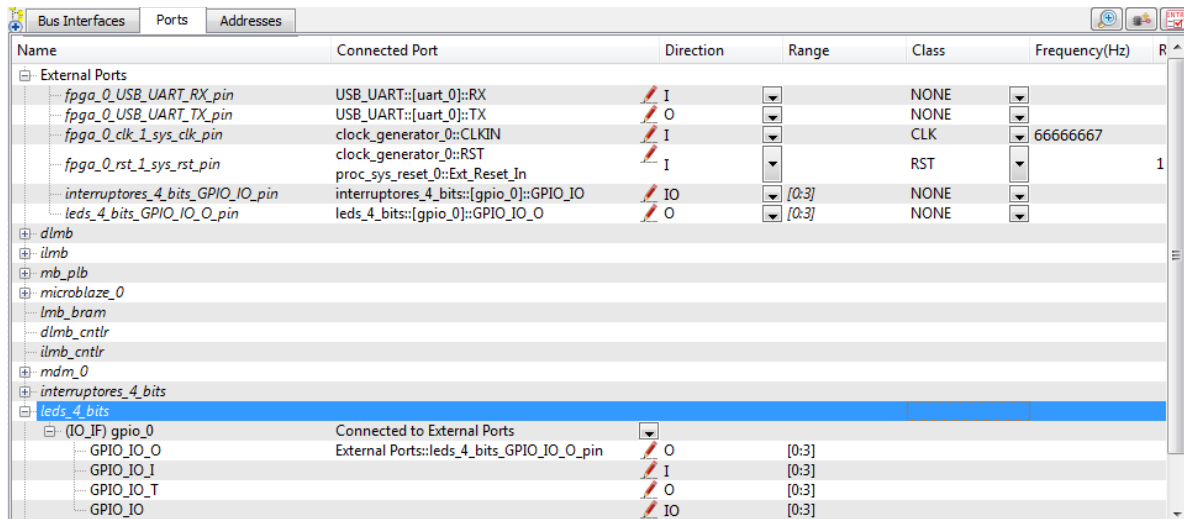
Conectamos las entradas del periférico a las señales interruptores_4_bits_GPIO_IO_pin y las definimos como externas en GPIO_IO.



El siguiente periférico a añadir y controlar, siguiendo los mismos pasos indicados anteriormente, serán los cuatro LEDs de la placa LX9. Le llamaremos leds_4_bits.

En este caso el tamaño del bus también será de 4 bits. Así mismo, definiremos el Channel 1 como FALSE al ser un periférico de salida.

Por último, conectamos las salidas del periférico (GPIO_IO_O) a las señales leds_4_bits_GPIO_IO_O_pin y las definimos como externas.



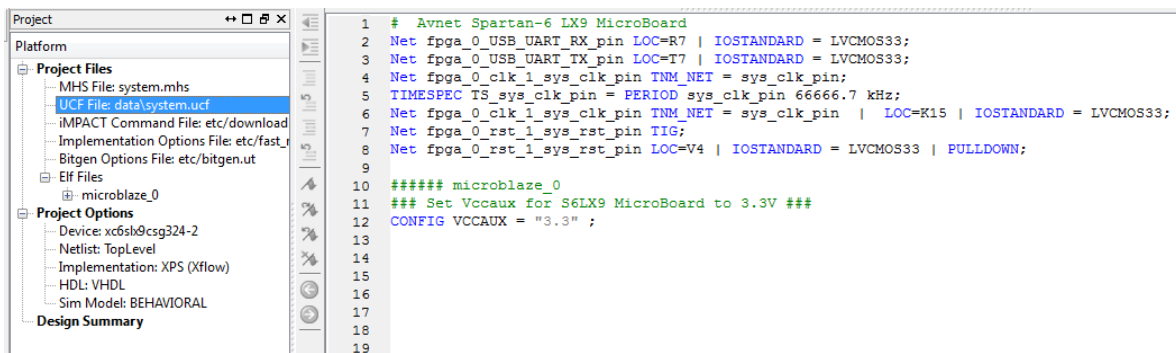
Hacemos los mismos pasos para agregar el pulsador de la tarjeta, en este caso vamos a usar uno de los PMODs externos de la placa. Colocaremos un cable en dicho pin y dándole alimentación o llevándolo a masa lo usaremos de pulsador. Le llamaremos pulsador_1_bit.

Estos nuevos periféricos no tienen asignada una dirección de memoria. Por tanto, debemos ir a la pestaña Addresses y bloquear las direcciones asignadas a los periféricos iniciales de nuestro sistema empotrado.

Pinchamos en Generate Addresses (icono que se encuentra en la pestaña Addresses en la parte superior derecha) y aparecerán las direcciones de nuestros nuevos periféricos.

Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name	Lock
microblaze_0's Address Map							
dlmb_cntlr	C_BASEADDR	0x00000000	0x00007FFF	32K	SLMB	dlmb	<input checked="" type="checkbox"/>
ilmb_cntlr	C_BASEADDR	0x00000000	0x00007FFF	32K	SLMB	ilmb	<input checked="" type="checkbox"/>
pulsador_1_bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
leds_4_bits	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
interruptores_4_bits	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb	<input type="checkbox"/>
USB_UART	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb	<input checked="" type="checkbox"/>
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb	<input checked="" type="checkbox"/>

Ahora debemos añadir los periféricos a nuestro archivo de restricciones para definir la asignación de los terminales externos. Abrimos el archivo system.ucf que se encuentra en la pestaña Project -> Project Files.



Para ello, debemos asignar los terminales de los periféricos. Podéis verlos en el datasheet de la PCB (xlx_s9_lx9_fpga_microboard-ug072711.pdf) y su definición (Avt_S6LX9_MicroBoard_UCF_110804.ucf) en el archivo anterior que tenéis colgado en el Aula Virtual.

2.6 User Interfaces

2.6.1 User LEDs

Four discrete "high-brightness, low Vf" LEDs are installed on the board and can be used to display the status of the internal logic. These LEDs are attached as shown below and are lit by forcing the associated FPGA I/O pin to a logic '1' and are off when the pin is either low (0) or not driven.

Net Name	Reference	FPGA Pin#
FPGA_GPIO_LED1	D2	P4
FPGA_GPIO_LED2	D3	L6
FPGA_GPIO_LED3	D9	F5
FPGA_GPIO_LED4	D10	C2

Table 11 – LED Pin Assignments

2.6.2 Four configurable FPGA user DIP switches (Tyco 1571983-4)

FPGA DIP switch	FPGA Pin#
FPGA_DIP1	B3
FPGA_DIP2	A3
FPGA_DIP3	B4
FPGA_DIP4	A4

Table 12 – FPGA DIP Switches

Please note that internal pulldowns are required for these pins.

2.5 User I/O and Expansion Connectors

2.5.1 Peripheral Module (PMOD)

Two 12-pin (2 x 6 female) Peripheral Module (PMOD) headers (J4, J5) are interfaced to the FPGA, with each header providing 3.3 V power, ground, and eight I/O's. These headers may be utilized as general-purpose I/Os or may be used to interface to PMODs. J4 and J5 are placed in close proximity (0.9"-centers) on the PCB in order to support dual PMODs. Table 9 and Table 10 provide the connector and FPGA pinout. For Digilent PMODs see: <http://www.digilentinc.com/pmmods>

FPGA Pin #	I/O Signal	Connector Pin #	Connector Pin #	I/O Signal	FPGA pin #
H12	FPGA_PMOD2_P1	1	7	FPGA_PMOD2_P7	K12
G13	FPGA_PMOD2_P2	2	8	FPGA_PMOD2_P8	K13
E16	FPGA_PMOD2_P3	3	9	FPGA_PMOD2_P9	F17
E18	FPGA_PMOD2_P4	4	10	FPGA_PMOD2_P10	F18
-	GND	5	11	GND	-
-	+3.3V_LS1	6	12	+3.3V_LS1	-

Table 9 – Peripheral Module Connections – J4

Observar bien la figura siguiente para saber como hacer un '0' y un '1' con el PMOD usando sus pines de 3V3 y GND.

El nombre que debemos poner a los terminales debe coincidir con el que aparece en la sección Ports del archivo system.mhs. Mirar como se definen los pines para saber cual es el primero, el segundo...

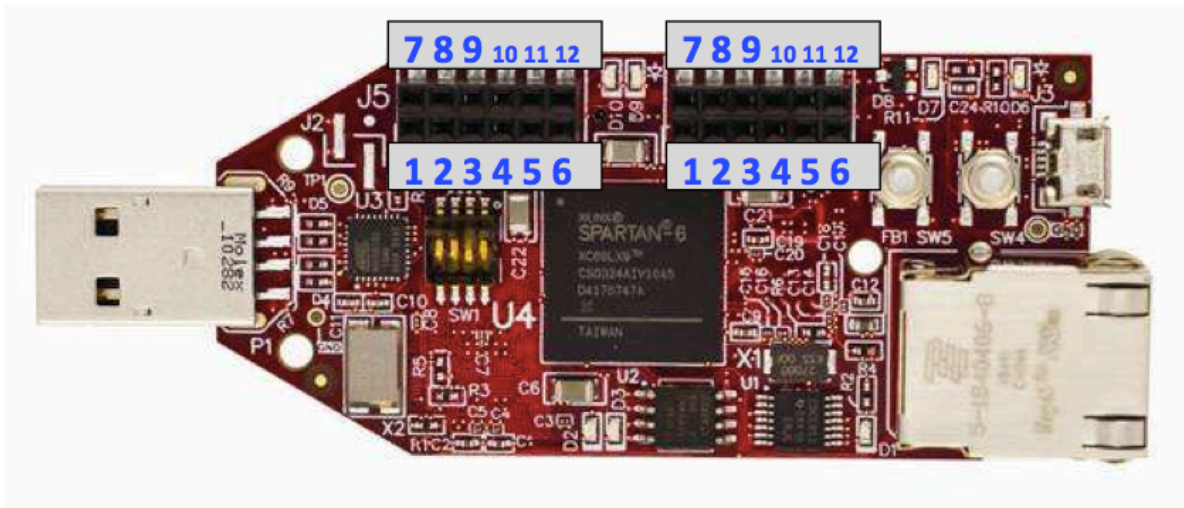


Figure 8 – PMOD Connector Pinout

Ahora debemos implementar de nuevo el hardware para añadir a nuestro sistema empujado los nuevos periféricos y comprobar que no hay errores (Hardware -> Generate Bitstream)

Para poder usar estos periféricos y comprobar que nuestra aplicación funciona correctamente, es necesario crear una nueva aplicación software. Las opciones referidas al desarrollo del software para el sistema empujado son accesibles a través de la opción Project -> Export Hardware Design to SDK...

En SDK seguimos los pasos de los anteriores Ejercicios para crear una Board Support Package referida al nuevo hardware creado y un archivo de Proyecto en C donde escribir nuestro código.

Añadimos el fichero fuente pulsando el botón derecho sobre el nombre de nuestro proyecto y seleccionamos New->Source File, se indica el nombre del proyecto (uso_periféricos), el nombre del archivo (control_periféricos.c) y el lenguaje que se va a emplear para su programación (C).

Debemos realizar un programa que lea el estado de los interruptores cuando “apretemos” el pulsador y muestre dicho valor en los LEDs y en Terminal. El valor debe permanecer fijo hasta que modifiquemos el mismo en los interruptores y volvamos a “apretar” el pulsador.