MÁSTER OFICIAL EN INGENIERÍA ELECTRÓNICA

SISTEMAS INTEGRADOS

BLOQUE 5: Mercado de los Sistemas Integrados

Curso 2013-2014

José Torres

Raimundo García

Julio Martos

Jesús Soret

Adrián Suárez

Pedro A. Martínez

Abraham Menéndez

DSDC Diseño de Sistemas Digitales y de Comunicaciones

VNIVERSITAT DE VALÈNCIA Escola Tècnica Superior d'Enginyeria

SISTEMAS INTEGRADOS

Tema 5.- Mercado de los Sistemas Integrados

Máster Oficial en Ingeniería Electrónica
Curso 2013-14

VNIVERSITAT DE VALÈNCIA Escola Tècnica Superior d'Enginyeria

Mercado de los Sistemas Integrados

Índice

- Fabricantes FPGAs con Sistemas Integrados.
- Otros Sistemas Integrados.
- Ferias, revistas, distribuidores y congresos.
- Ejemplos y vídeos.
- Módulos complementarios.
- Empresas y aplicaciones.

Fabricantes FPGAs con Sistemas Integrados

- Xilinx es la mayor empresa en investigación y desarrollo de chips tipo FPGA y SoC de altas prestaciones.
- Sus competidores son Altera, Lattice y Actel.
- En Microprocesadores software embebidos tenemos:
  - Xilinx: Spartan y Virtex (PicoBlaze 8-bits y MicroBlaze 32-bits)
  - Altera: Cyclone y Stratix (Nios 16-bits y Nios II-32 bits en versiones /f (fast) /s (standard) /e (economy)), Cyclone III (Cortex-M1 ARM, Freescale ColdFire, Intel Atom)
  - Lattice: MachXO, iCE40, LatticeECP2/3 y LatticeXP2 (LatticeMico8 8-bits), LatticeECP2/3 y LatticeXP2 (LatticeMico32 32-bits)
  - Actel (Microsemi): Fusion (Cortex-M1 ARM)

Mercado de los Sistemas Integrados

Fabricantes FPGAs con Sistemas Integrados

- Microprocesadores software embebidos

LatticeMico32

PicoBlaze™

Nios® II

Cortex
Intelligent Processors by ARM

MicroBlaze

LatticeMico8™

Embedded CPU soft cores suitable for FPGA

CPU core	Architecture	Bits	License	Pipeline depth	Cycles per instruction[1]	MMU[2]	MUL[3]	FPU[4]	Area (LEs[5])	Comments
S1 Core	SPARC-v9	64	Open-source (GPL)	6	1	+	+	+	37000 - 60000	Single-core version of UltraSPARC T1
LEON3	SPARC-v8	32	Open-source (GPL)	7	1	+	+	+	3500	
LEON2	SPARC-v8	32	Open-source (LGPL)	5	1	+	+	ext	5000	Unmaintained in favor of LEON3
OpenRISC 1200	OpenRISC 1000	32	Open-source (LGPL)	5	1	+	+	-	6000	
MicroBlaze	MicroBlaze	32	Proprietary	3, 5	1	opt	opt	opt	1324	Limited to Xilinx devices
aeMB	MicroBlaze	32	Open-source (LGPL)	3	1	-	opt	-	2536	Open-source clones of MicroBlaze
OpenFire	MicroBlaze	32	Open-source (MIT)	3	1	-	opt	-	1928	
Nios II/f	Nios II	32	Proprietary	6	1	+	+	opt	1800	Limited to Altera devices
Nios II/s	Nios II	32	Proprietary	5	1	-	+	opt	1170	
Nios II/e	Nios II	32	Proprietary	no	6	-	-	opt	390	
LatticeMico32	LatticeMico32	32	Open-source	6	1	-	opt	-	1984	Not limited to Lattice devices (can be used elsewhere)
Cortex-M1	ARMv6	32	Proprietary	3	1	-	+	-	2600	
DSPuva16	DSPuva16	16	Open-source	no	4	-	+	-	510	
PicoBlaze	PicoBlaze	8	Proprietary, zero-cost	no	2	-	-	-	192	Limited to Xilinx devices
PacoBlaze	PicoBlaze	8	Open-source (BSD)	no	2	-	-	-	204	An open-source clone of PicoBlaze
LatticeMico8	LatticeMico8	8	Open-source	no	2	-	-	-	200	Not limited to Lattice devices (can be used elsewhere)

Mercado de los Sistemas Integrados

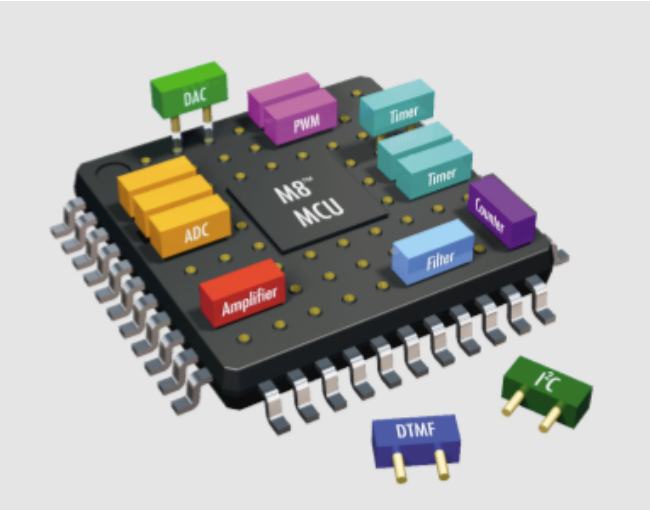
Fabricantes FPGAs con Sistemas Integrados

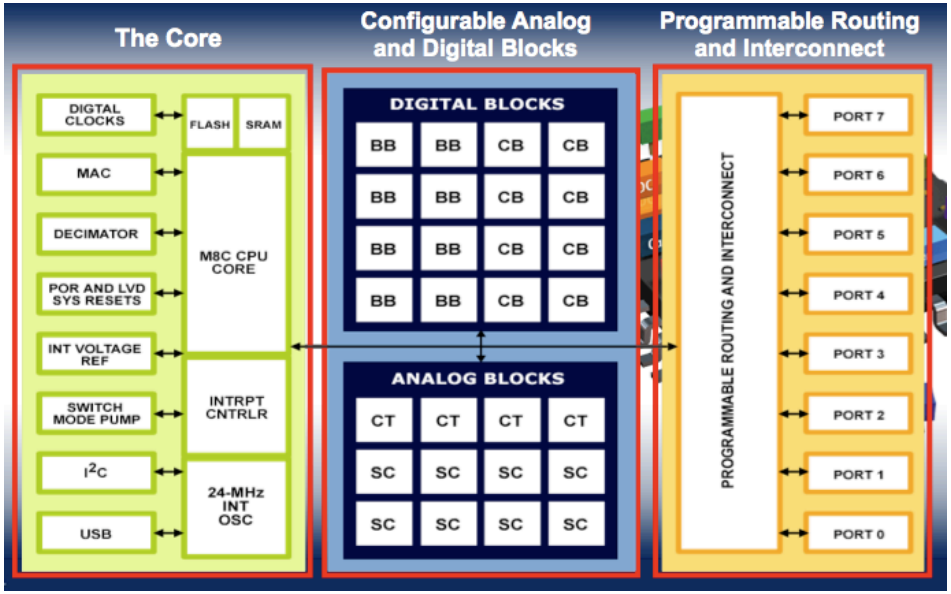
- En Microprocesadores hardware embebidos tenemos:
  - Xilinx: Virtex (PowerPC), Zynq (Cortex-A9 ARM)
  - Altera: Arria V y Cyclone V (Cortex-A9 ARM)
  - Lattice: No tiene
  - Actel (Microsemi): SmartFusion (Cortex-M3 ARM)

Mercado de los Sistemas Integrados

Otros Sistemas Integrados

- Existen otros fabricantes dedicados a SoCs más específicos:
  - Cirrus Logic: Familia CS470xx, dedicados a Audio, integran un DSP en lugar de un Microprocesador.
  - Cypress: Familias PSoC 1-3-5.





Mercado de los Sistemas Integrados

Revistas y Distribuidores

- http://www.embedded.com
- http://www.silica.com/products/technology/pld-fpga.html
- http://www.xilinx.com/publications/xcellonline/index.htm
- http://www.digilentinc.com
- http://es.mouser.com
- http://www.digikey.es
- http://avnetexpress.avnet.com

Mercado de los Sistemas Integrados

Ferias y Congresos

- Embedded World, evento que se celebra a final del mes de febrero en Nuremberg. Se trata de la feria más importante del mundo para Sistemas Embebidos.
- Este año se registraron más de 26.000 visitantes y más de 850 expositores. http://www.embedded-world.de/en/
- http://www.vlsidesignconference.org
- http://www.esweek.org

Mercado de los Sistemas Integrados

Ejemplos y Vídeos

- http://www.xilinx.com/publications/archives/xcell/Xcell51.pdf
- http://www.atomium.es/las-palmas/modulador-digital-tecatel-dim2/
- http://www.youtube.com/XilinxInc
- http://johnnylee.net/projects/wii/
- http://www.waltergallegos.com/pdf/xjtag_tecnobit_estudio-de-caso.pdf
- http://www.waltergallegos.com/pdf/xjtag_kerajet_estudio-de-caso.pdf
- http://www.ni.com/academic/esa/embedded.htm

Módulos Complementarios

- Enclustra: Mars ZX3 Modulo DIMM
  http://www.enclustra.com/en/home/
- Net Module: Zynq4 5 Ethernet
  http://www.netmodule.com/products/sbc-eval-sys/zynq4ethernet-solution.html
- Trenz Electronics: Mini módulo
  http://www.trenz-electronic.de/products/fpga-boards/trenz-electronic/te0720-zynq.html
- AF Inventions: Stereovision System
  http://www.af-inventions.de/en/fpga-solutions/gimme2/
- TED Electronics: Kit desarrollo
  http://solutions.inrevium.com/products/base/zynq/tb-7z-020-emc.html

Mercado de los Sistemas Integrados

Empresas y Aplicaciones

- Empresas:
  - Alcatel
  - Dycec
  - Indra
  - Kerajet
  - Matrix
  - Futurasmus
  - Fermax
  - Domoval
  - Diteco
  - Miniatec
  - Oncovision
  - IZ Ingenieros
  - ETRA
  - Securitas Direct
  - Ford
  - Telefónica
  - SAB
  - Biodit
  - Ingeniería Urbótica
  - Tecnobit
  - SAES
  - CERN
  - Lepton
  - Marvell
  - Power Electronics
  - …

Mercado de los Sistemas Integrados

Empresas y Aplicaciones

- Aplicaciones:
  - Alarmas
  - Alimentación eléctrica
  - Arcos de seguridad
  - Climatización de autobuses
  - Comunicaciones por fibra óptica
  - Conducción automática de trenes
  - Control industrial
  - Control de instalaciones eléctricas
  - Electrónica espacial
  - Electrónica submarina
  - Domótica
  - Detectores radiación
  - Ensayo de materiales
  - Equipos de transmisión de TV
  - Fabricación de azulejos y cerámicos
  - Equipos de medicina y radiología
  - Navegación GPS
  - Telefonía móvil
  - …

Mercado de los Sistemas Integrados

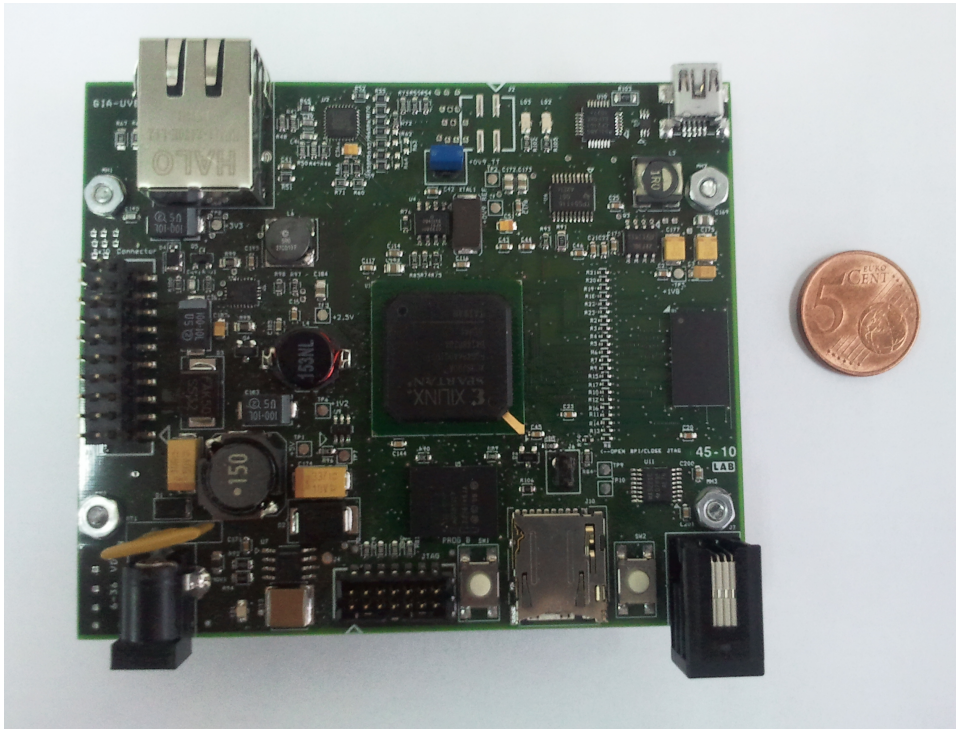
Empresas y Aplicaciones

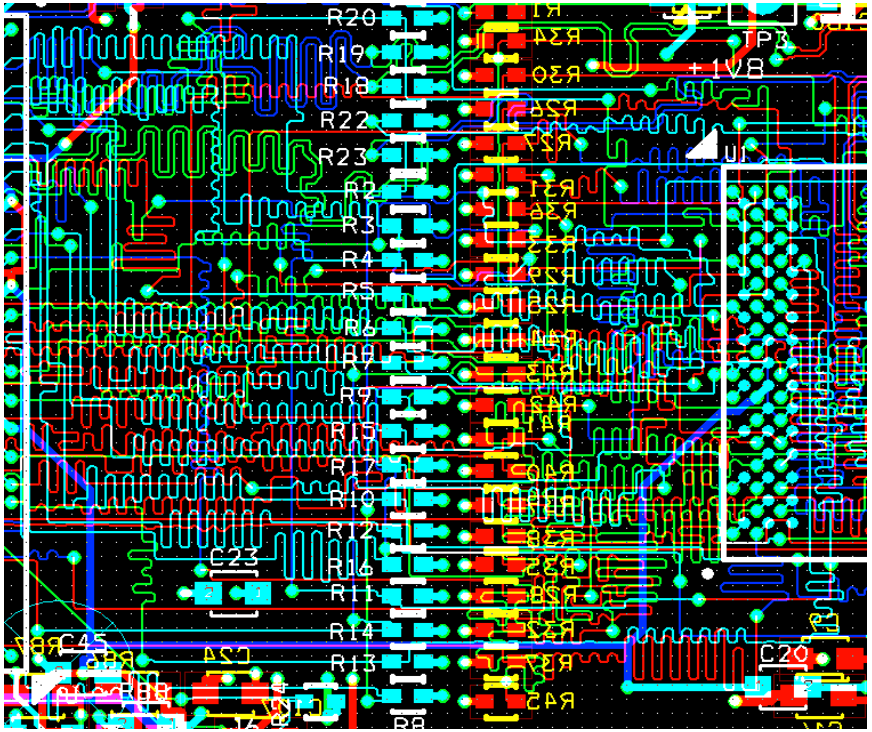
- Aplicaciones DSDC:

Mercado de los Sistemas Integrados

Empresas y Aplicaciones

- Aplicaciones DSDC:

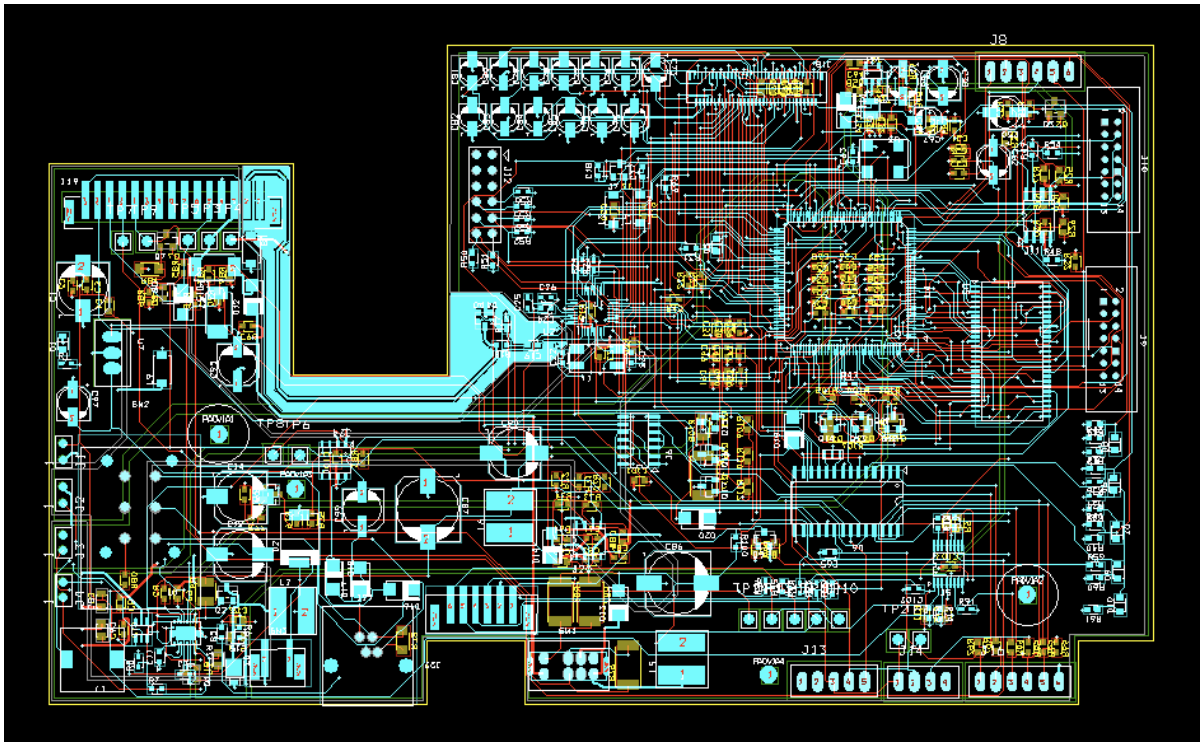




Mercado de los Sistemas Integrados

Empresas y Aplicaciones

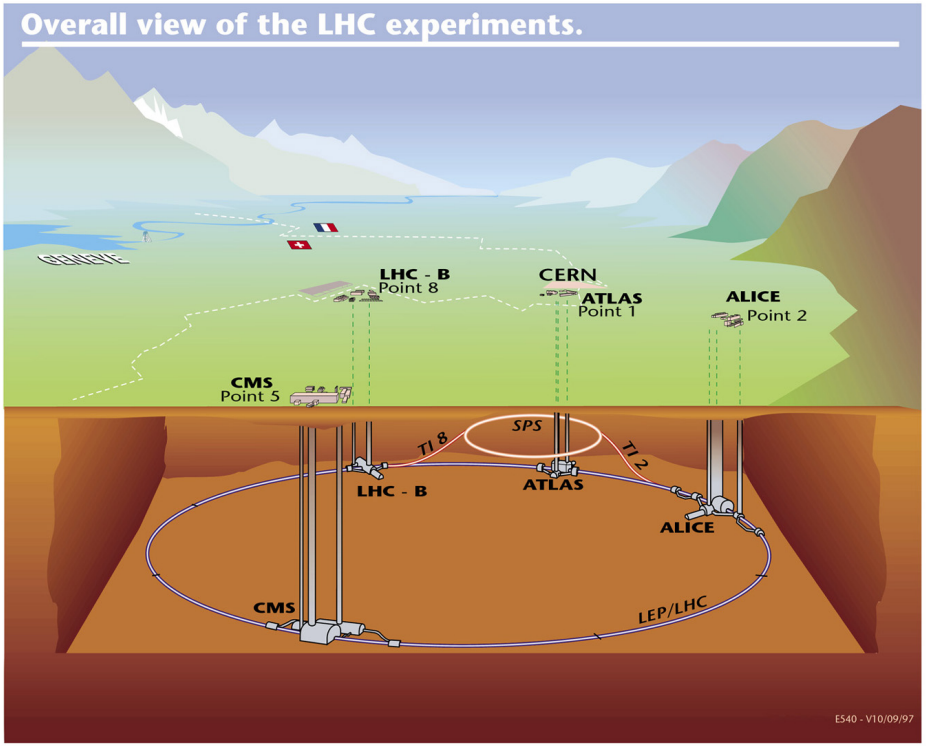
- Aplicaciones DSDC:

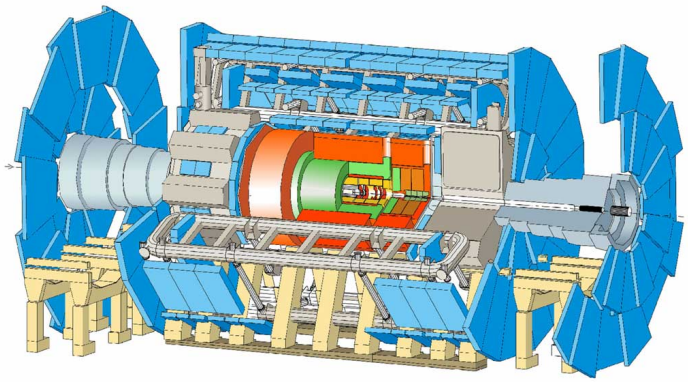


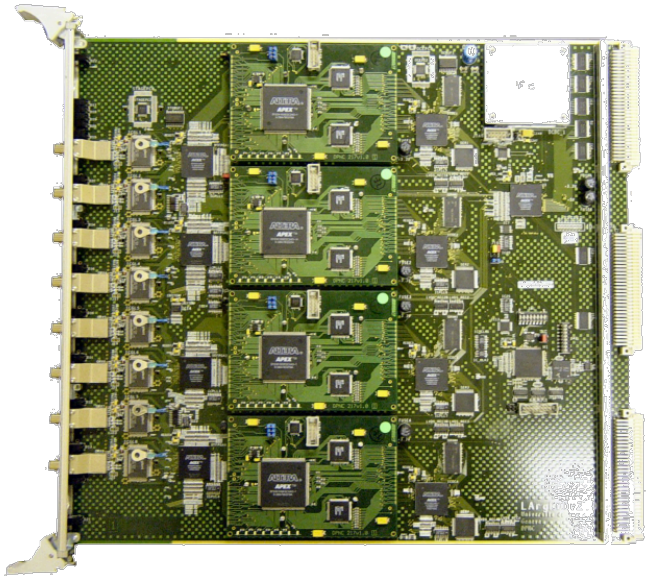
Mercado de los Sistemas Integrados

Empresas y Aplicaciones

- Aplicaciones DSDC:

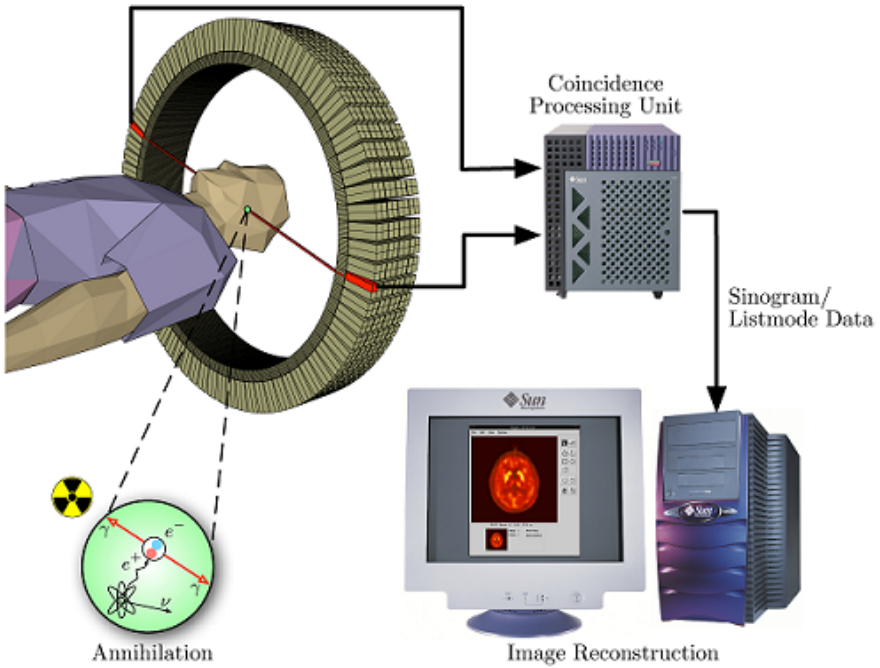


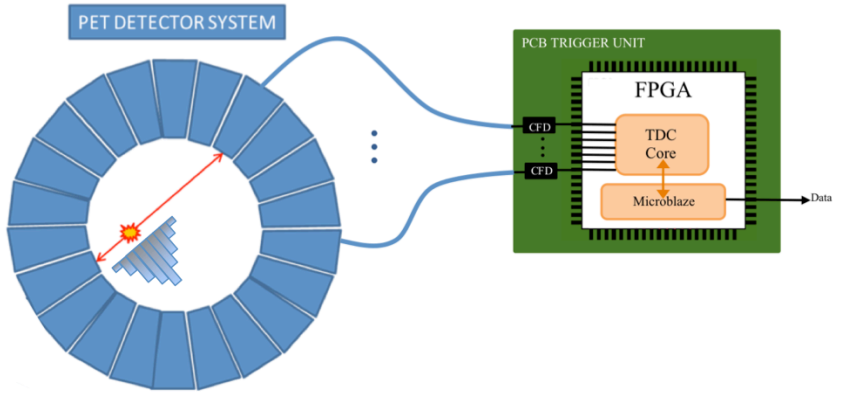


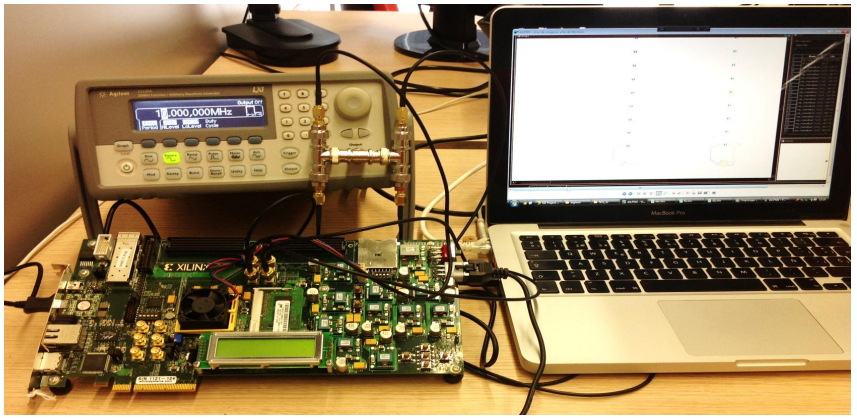


Empresas y Aplicaciones

- Aplicaciones DSDC:



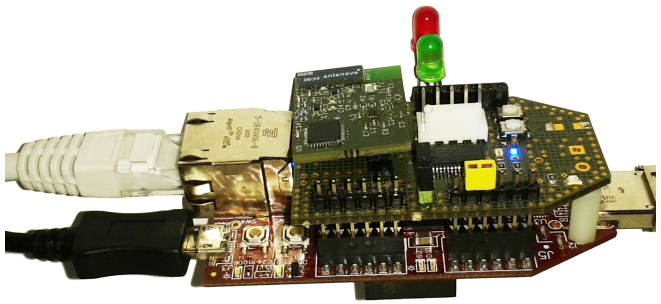


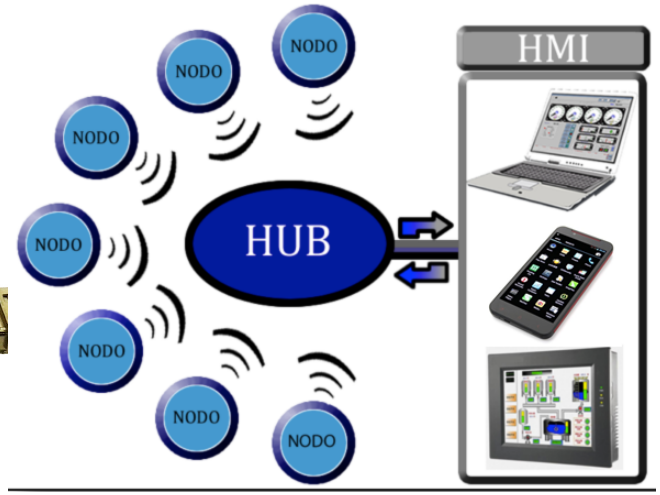


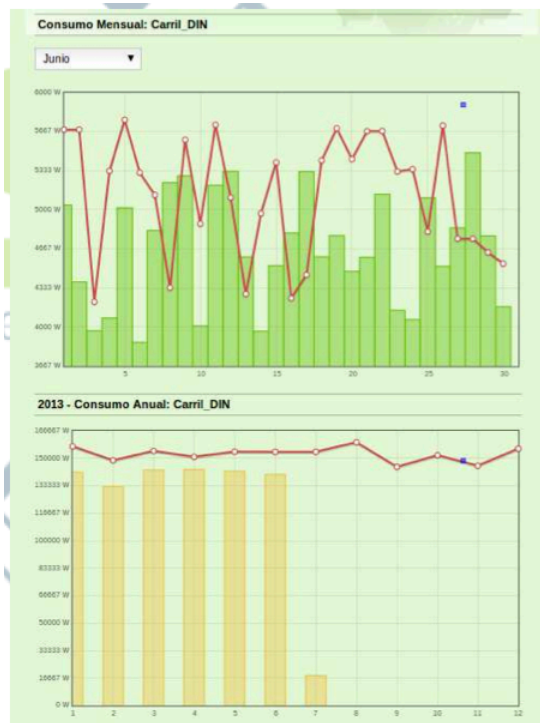
Mercado de los Sistemas Integrados

Empresas y Aplicaciones

- Aplicaciones DSDC:
- http://dsdc.no-ip.biz







Ejercicio

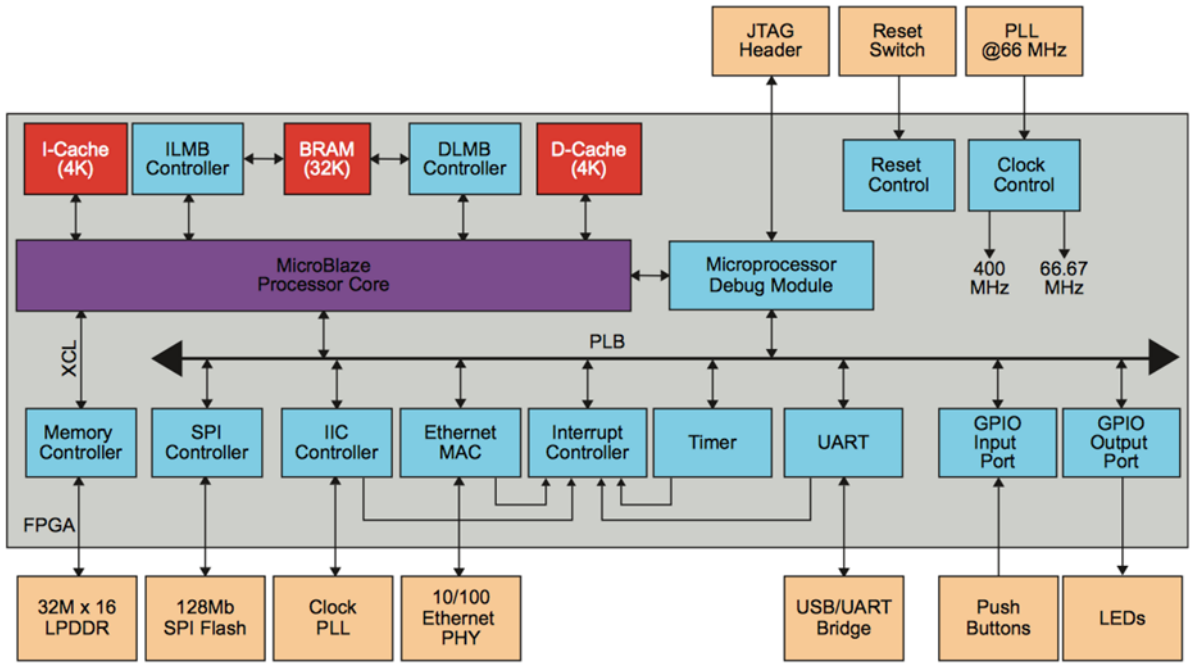
8

Creación de un Periférico de Usuario

Una de las ventajas de MicroBlaze es el número de periféricos de los cuales disponemos en el IP Catalog. Si nuestro periférico no aparece en dicha lista, podemos crearlo con una aplicación que incluye EDK.

Los objetivos de este ejercicio son la utilización del asistente para la creación e importación de periféricos, la exploración de las plantillas y de la estructura de directorios creados por el asistente, la modificación de dichas plantillas para añadir la funcionalidad del periférico y poder simularlo con Foundation ISE, la forma de añadir el nuevo periférico a un diseño ya existente y la creación de una aplicación software para comprobar la funcionalidad del periférico.

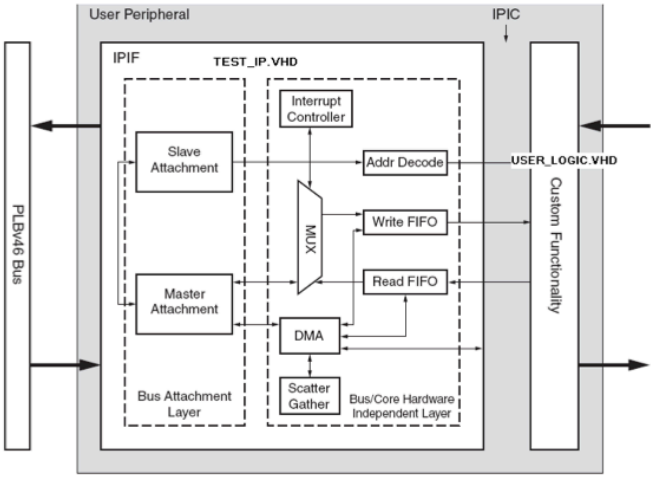
El diagrama de bloques que tendrá esta aplicación es el siguiente.



Creación sistema empotrado

MicroBlaze presenta una serie de librerías con periféricos que podemos conectar cómodamente a nuestro sistema empotrado. Cuando nuestra aplicación requiera de un periférico que no exista en dichas librerías, podremos usar la aplicación de Creación e Importación de Periféricos que incorpora EDK.

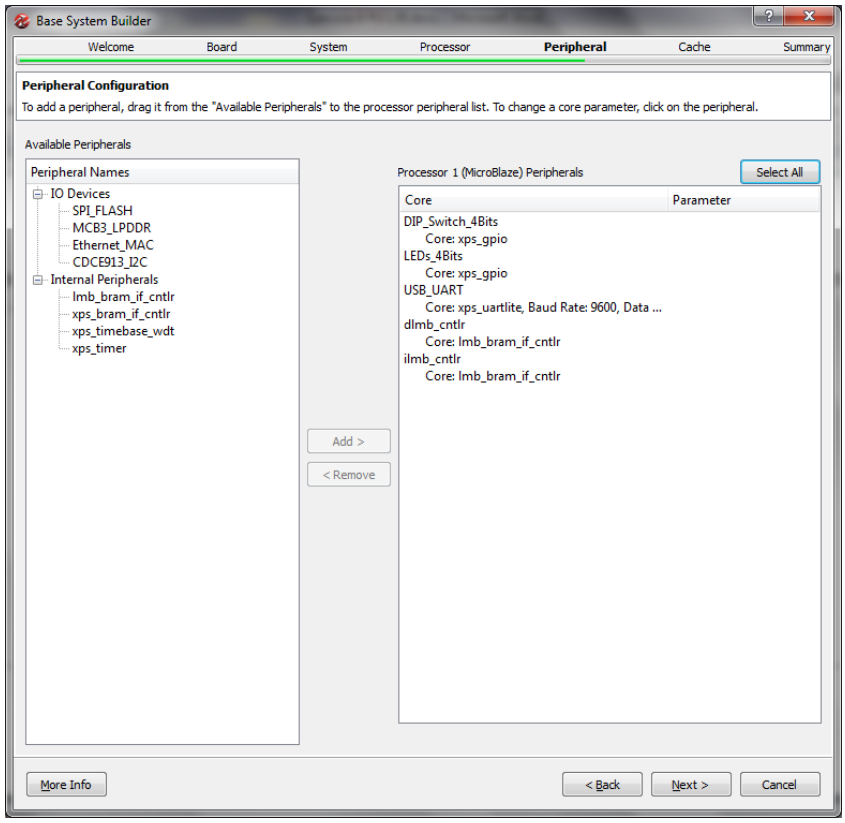
MicroBlaze, en la versión 10, se conecta a los periféricos mediante el Bus PLB. Es necesario, por tanto, conectar nuestro módulo-periférico a dicho bus.



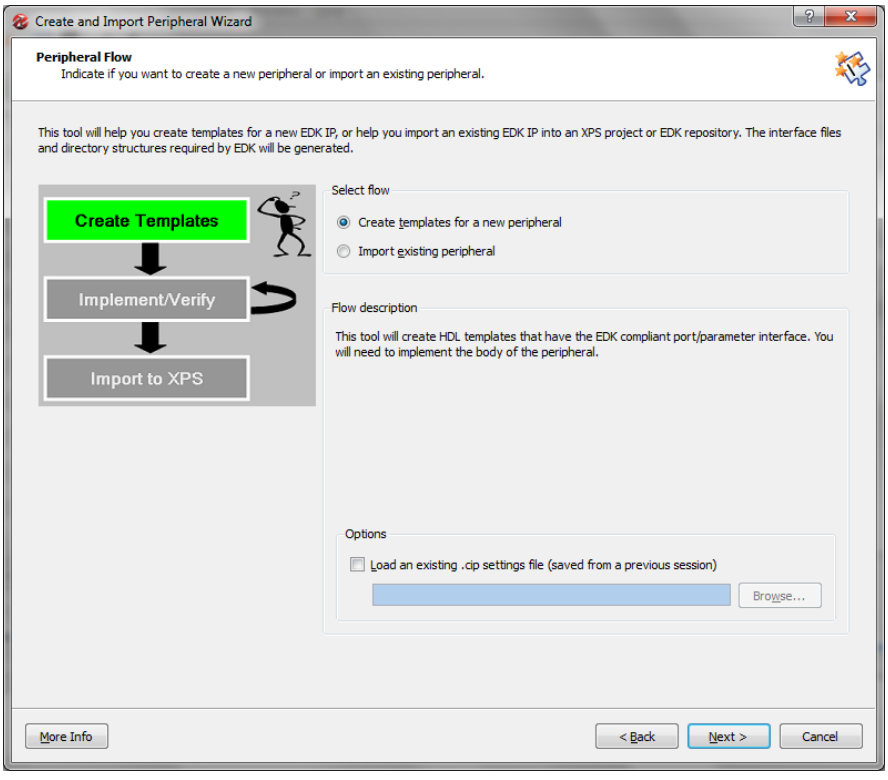
EDK usa la librería IPIF (Intellectual Property Interface) para implementar la funcionalidad sobre los periféricos de usuario, nos permite de una manera rápida y sencilla conectar nuestro diseño con el Bus PLB. Para ello, usa un protocolo de bus simplificado llamado IPIC (IP Interconnect) el cual nos permite acceder al Bus PLB controlando sólo unas cuantas líneas.

Ahora ya estamos en disposición de empezar nuestro diseño. Siguiendo los pasos de ejercicios anteriores, creamos un nuevo proyecto con el nombre Ejercicio_8. 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 USB_UART, así como los LEDS y los Switches. Eliminaremos el resto de periféricos, tal y como se muestra en la siguiente figura:

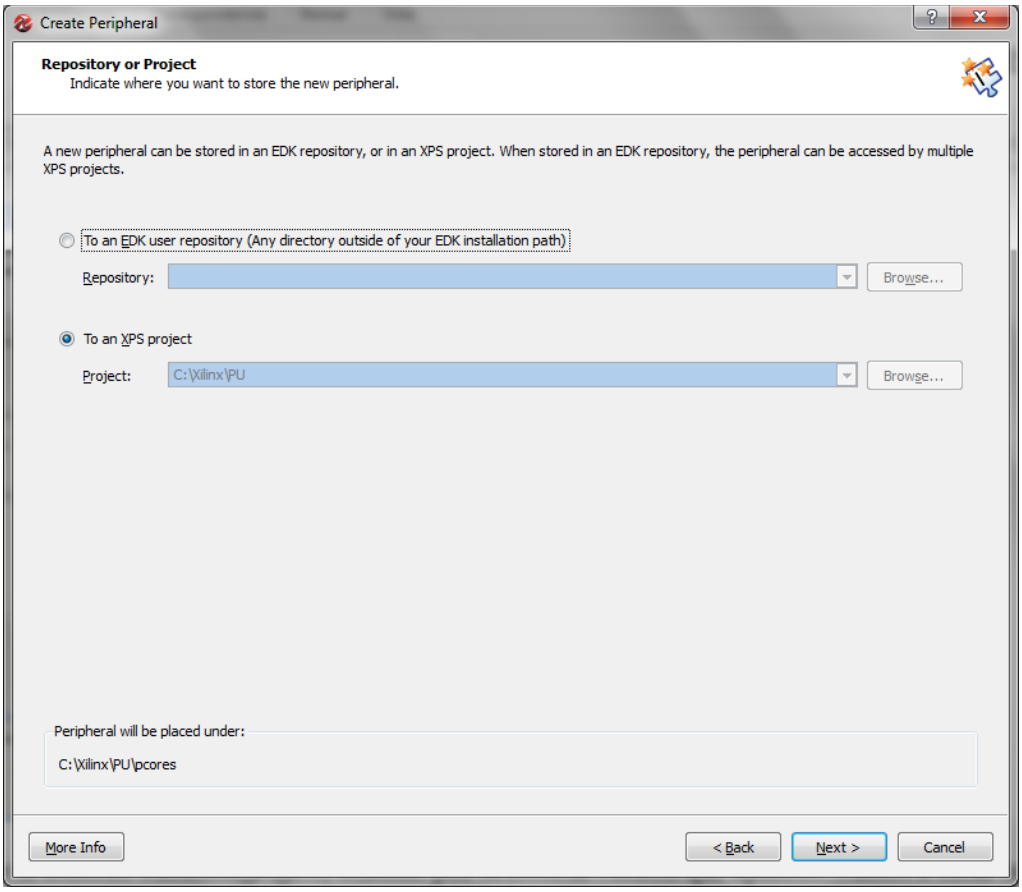


Una vez generado nuestro hardware, vamos a pasar a crear un nuevo periférico. Para ello debemos utilizar la opción Hardware -> Create or Import Peripheral. Con esto se inicia el asistente para la creación de nuevos periféricos de usuario. Seleccionamos la primera opción Create templates for a new peripheral.

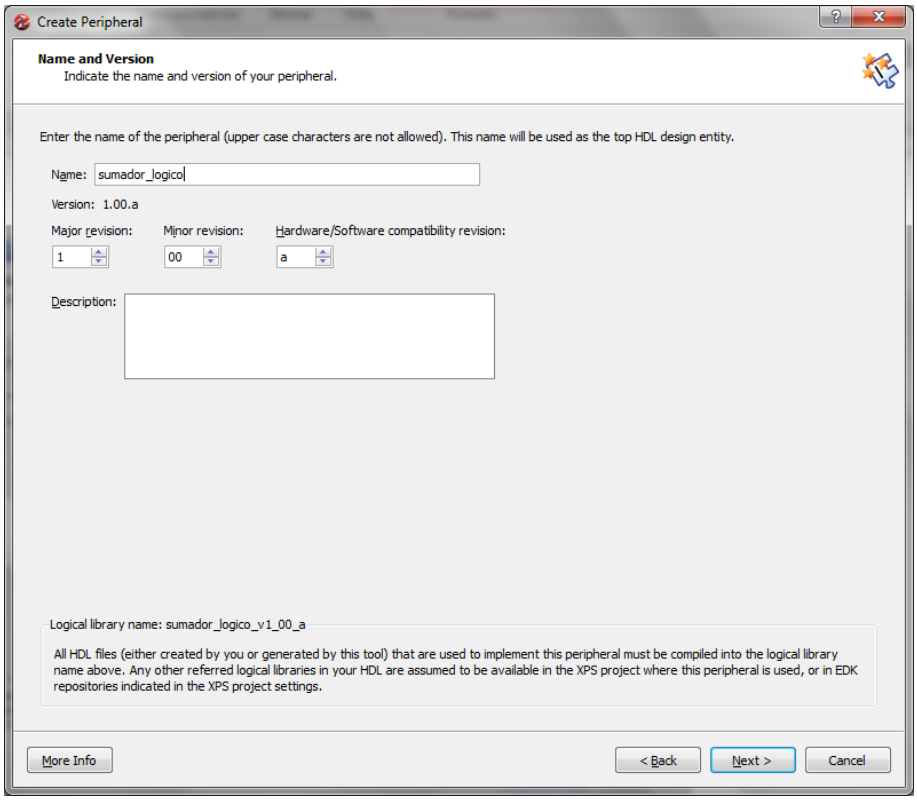


Los archivos de descripción del periférico se pueden incluir dentro del directorio del proyecto o en otro directorio de usuario (user repository). Esta última opción permite a distintos proyectos acceder a los periféricos de usuario.

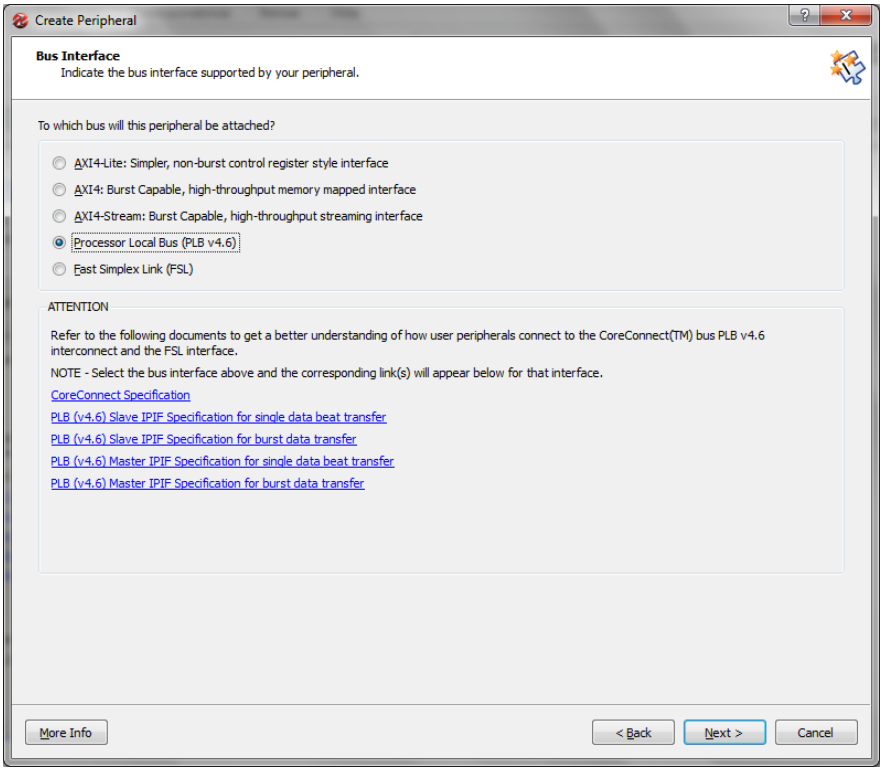
Por tanto, esta opción resulta adecuada para periféricos genéricos como podría ser el circuito controlador de un LCD. En nuestro caso, vamos a hacer un sumador lógico que será específico de éste proyecto y escogeremos la primera opción.



Le damos el nombre sumador_logico, dejando por defecto la versión que aparece. Vamos a crear un bloque periférico propio que recibirá los valores de los Switches y realizará la suma lógica de los mismos, mostrando el resultado en el puerto serie.



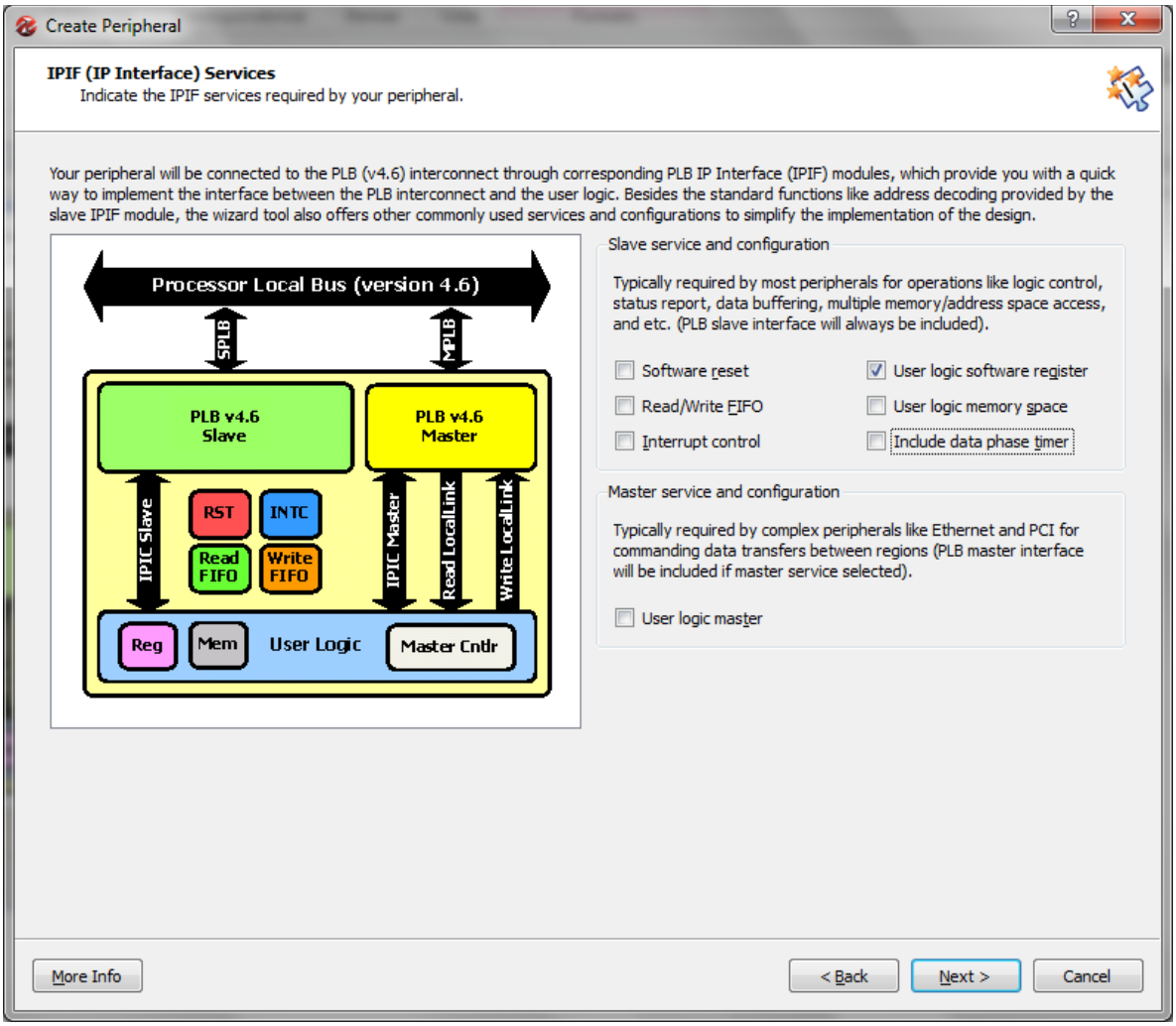
Seleccionamos el bus PLB (Processor Local Bus) para conectar el periférico al microprocesador. Los enlaces de este cuadro de diálogo nos permiten acceder a los archivos de descripción de los distintos interfaces de conexión a periféricos IPIF (Intellectual Property Interface). En nuestro caso vamos a crear un periférico esclavo conectado al bus PLB, por lo que su documentación corresponde al enlace PLB (v4.6).



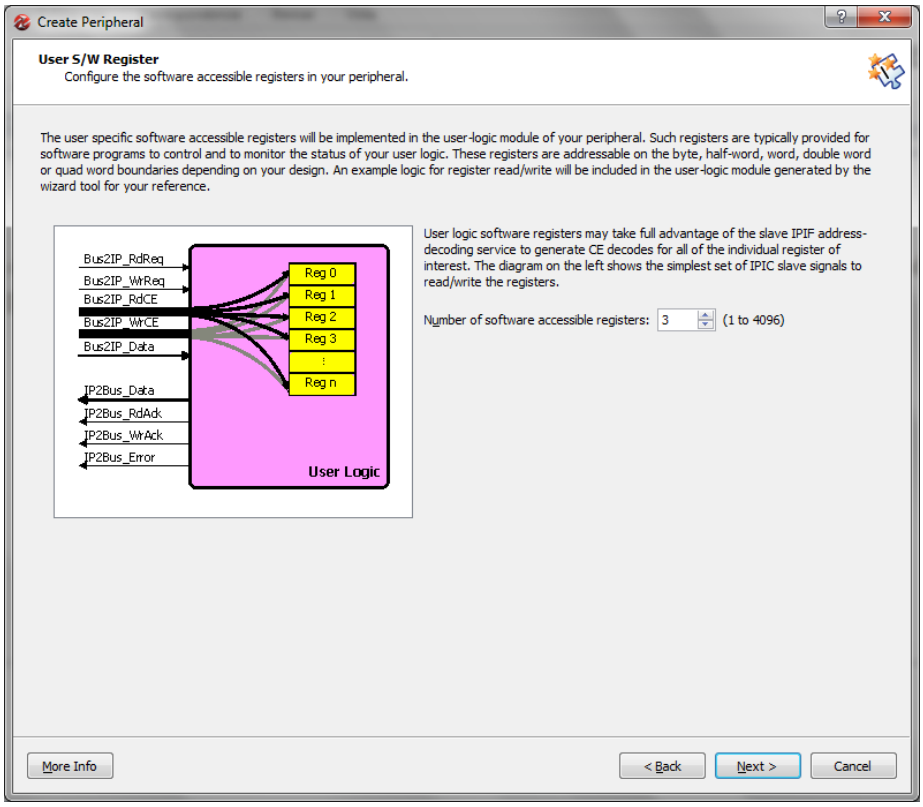
El nuevo cuadro de diálogo que aparece nos permite seleccionar los servicios IPIF que necesitamos que posea el interfaz de nuestro periférico.

En este caso, como se trata de un periférico simple, sólo necesitamos que disponga de la posibilidad de utilizar los registros de usuario accesibles por software (User logic software register). Con ello conseguimos, como podéis ver en la figura, almacenar los datos recibidos/enviados desde/hacia el microprocesador.

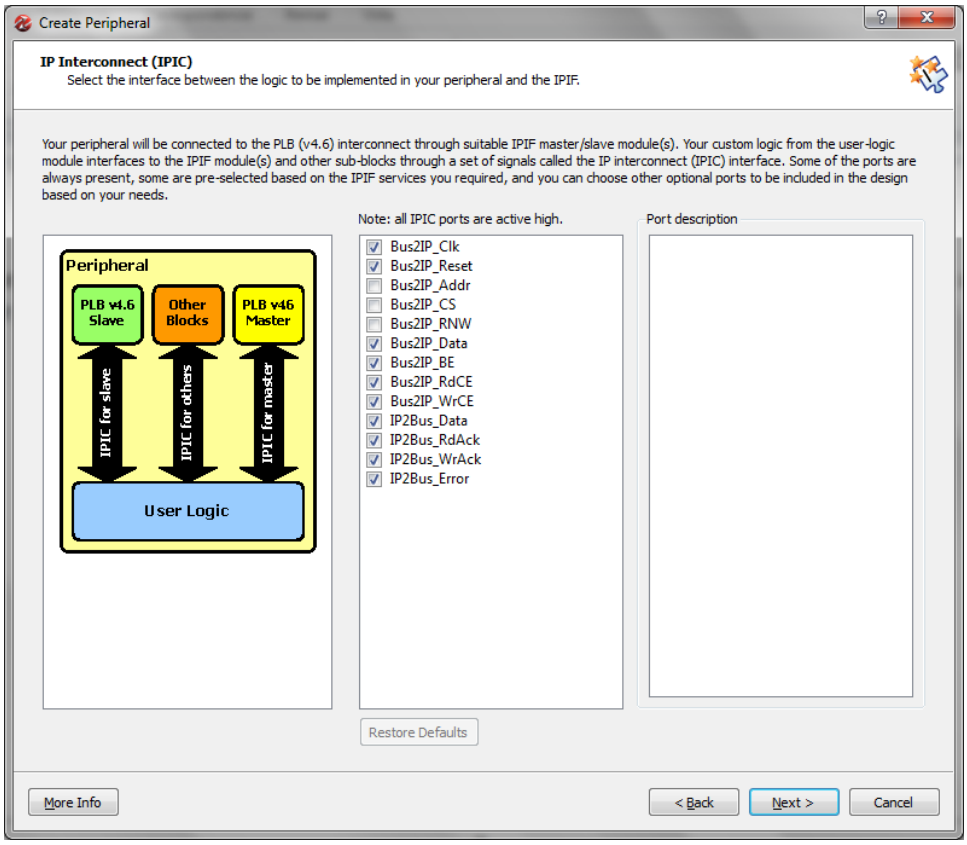
Podemos consultar la ayuda de estas opciones pinchando el botón More Info.



En el nuevo cuadro de diálogo, establecemos el ancho del bus de conexión del periférico esclavo con el bus PLB al valor que aparece por defecto, es decir, 32 bits. Finalmente, le decimos que vamos a utilizar 3 registros para la comunicación entre el periférico y MicroBlaze. Dos registros serán para cada uno de los sumandos, mientras que el tercero es donde se almacenará el resultado de la suma lógica.

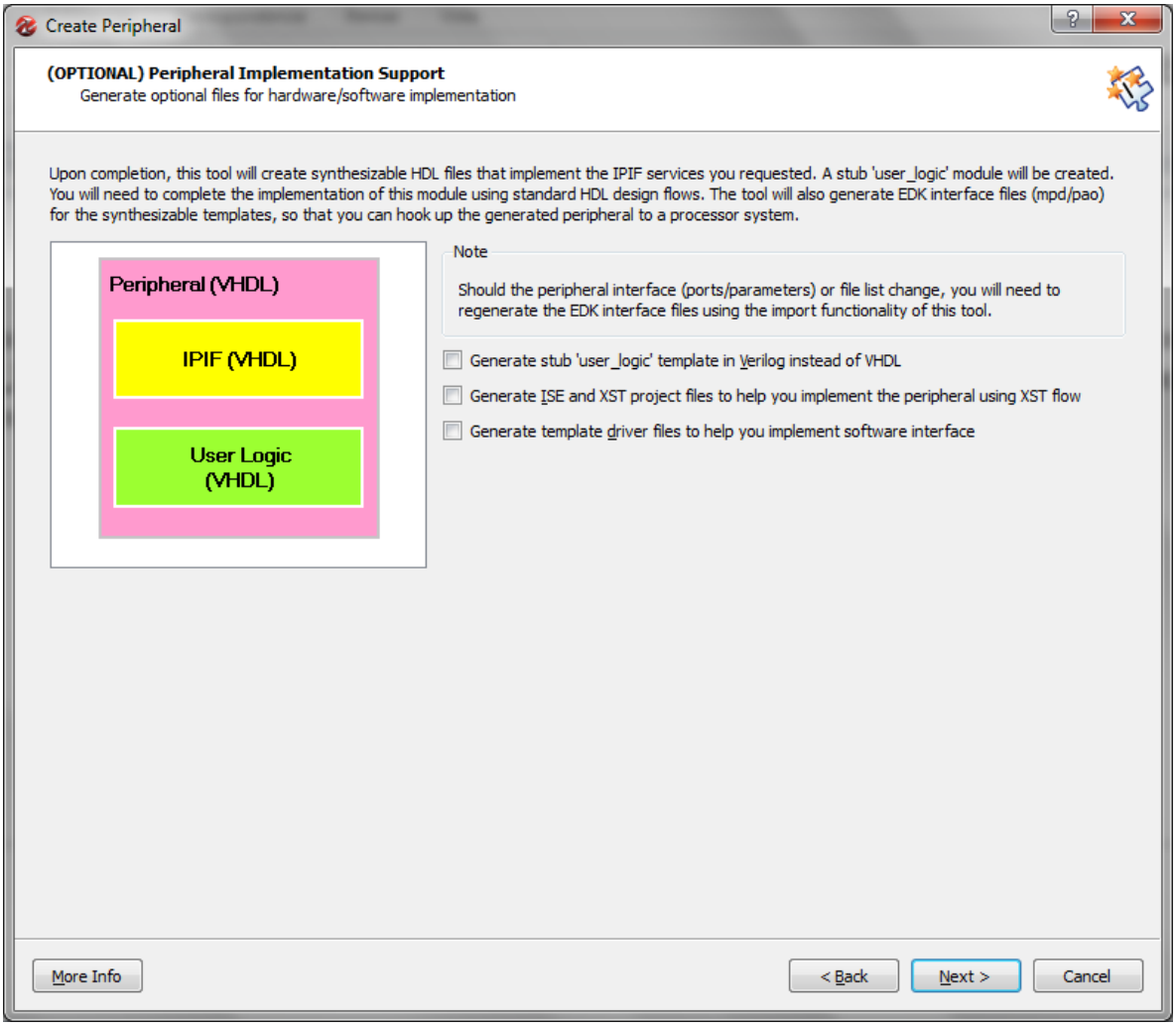


También definimos la conexión IPIC entre el bloque de usuario y el módulo esclavo de salida de datos. Dejamos por defecto las señales que aparecen.



Si disponemos de la herramienta ModelSim en su versión PE o SE (no es nuestro caso) podemos activar la opción de generar los ficheros necesarios para simular el periférico.

Finalmente, la opción "Generate template driver files…" seleccionada también por defecto nos puede ayudar a crear posteriormente el driver software del periférico para su manejo desde el microprocesador empotrado, de modo que la dejaremos activada.



Finalizamos el asistente y observamos como en la pestaña IP Catalog se ha creado un apartado denominado sumador_logico. No obstante, aún no podemos añadirlo al sistema empotrado puesto que no hemos definido su funcionalidad.

En la carpeta del ejercicio, la herramienta EDK ha creado las carpetas (drivers y pcores). La primera contiene los archivos relacionados con el driver software del periférico. La segunda almacena todos los archivos relacionados con el hardware del periférico.

Para cada periférico, la herramienta EDK genera unas carpetas equivalentes con el nombre del periférico correspondiente.

En el subdirectorio …\pcores\sumador_logico_v1_00_a\devl\projnav encontramos un proyecto de la herramienta ISE. Pinchando sobre él, abriríamos ISE y podríamos testear el hardware de nuestro periférico.

En el subdirectorio …\pcores\sumador_logico_v1_00_a\hdl\vhdl tenemos los archivos principales que definen nuestro periférico. Uno es sumador_logico.vhd, en el cual podemos añadir los terminales externos adicionales que necesite nuestro periférico (no es nuestro caso, éste periférico no tiene comunicación con el exterior).

Los terminales necesarios para conectar el periférico con el microprocesador ya han sido definidos automáticamente por la herramienta EDK. Se recomienda analizar la estructura de este archivo.

El otro archivo es user_logic.vhd, debemos definir la funcionalidad de nuestro periférico en éste archivo. Para realizar la descripción, podemos aprovechar las descripciones de los registros del periférico que incluye el propio archivo.

Por tanto, en este caso, dado que vamos a hacer por software todo el control del sumador añadimos las líneas:

- signal sum : std_logic_vector(0 to C_SLV_DWIDTH-1);
- signal c : std_logic_vector(0 to 32);

```
127
128  architecture IMP of user_logic is
129
130    --USER signal declarations added here, as needed for user logic
131
132    ---------------------------------------------
133    -- Signals for user logic slave model s/w accessible register example
134    ---------------------------------------------
135    signal slv_reg0                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
136    signal slv_reg1                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
137    signal slv_reg2                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
138    signal slv_reg_write_sel              : std_logic_vector(0 to 2);
139    signal slv_reg_read_sel               : std_logic_vector(0 to 2);
140    signal slv_ip2bus_data                : std_logic_vector(0 to C_SLV_DWIDTH-1);
141    signal slv_read_ack                   : std_logic;
142    signal slv_write_ack                  : std_logic;
143    signal sum                            : std_logic_vector(0 to C_SLV_DWIDTH-1);
144    signal c                              : std_logic_vector(0 to 32);
145
146
147  begin
148
```

Ahora añadimos la señal sum al proceso de lectura, así podemos manejar el tercer registro para leer dicha suma. Fijaros en la siguiente imagen y modificarlo.

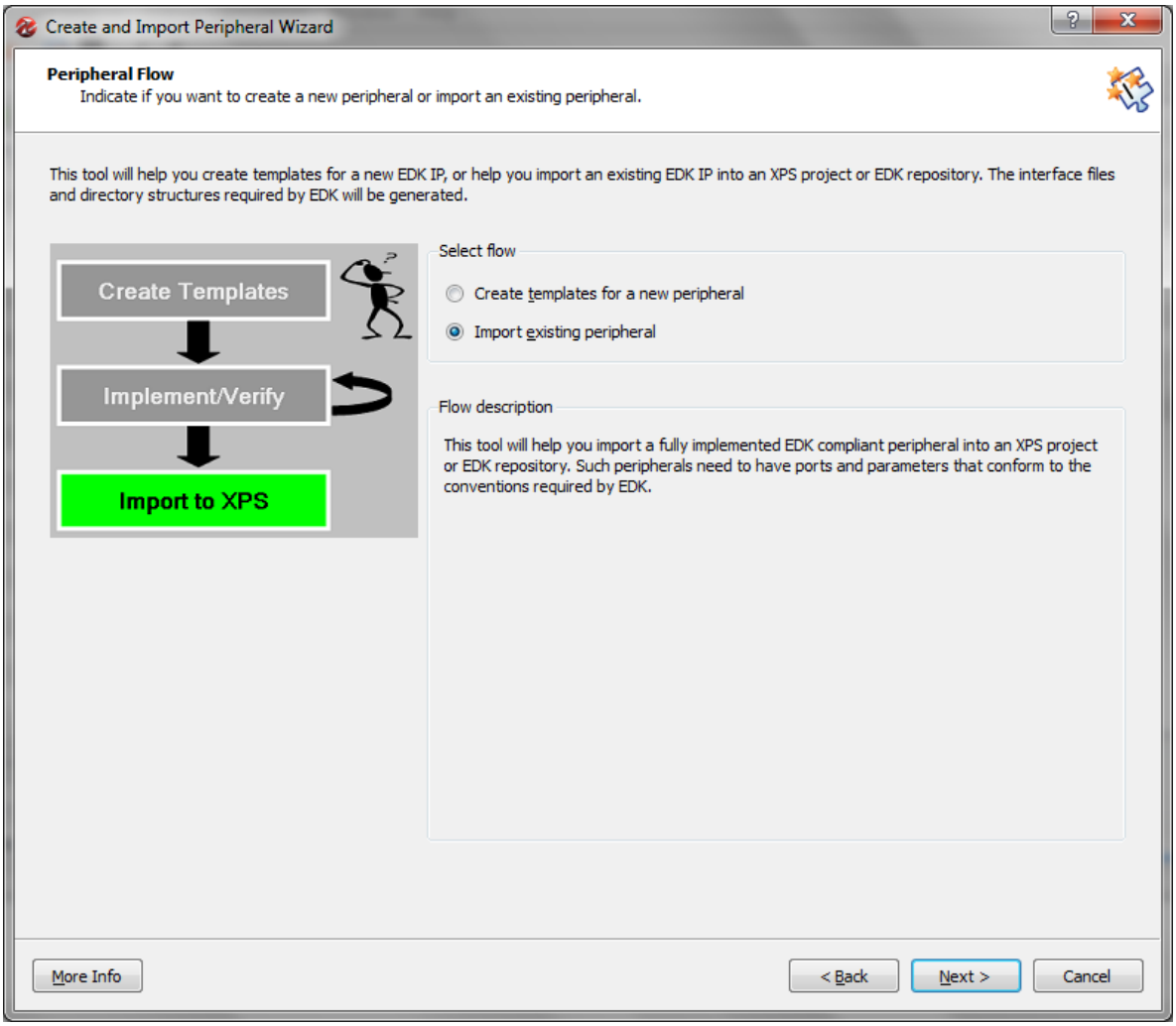
```
209
210    -- implement slave model software accessible register(s) read mux
211    SLAVE_REG_READ_PROC : process( slv_reg_read_sel, slv_reg0, slv_reg1, slv_reg2, sum ) is
212    begin
213
214      case slv_reg_read_sel is
215        when "100" => slv_ip2bus_data <= slv_reg0;
216        when "010" => slv_ip2bus_data <= slv_reg1;
217        when "001" => slv_ip2bus_data <= sum;
218        when others => slv_ip2bus_data <= (others => '0');
219      end case;
220
221    end process SLAVE_REG_READ_PROC;
```

Por último, añadimos el proceso de suma que queremos que nuestro periférico realice.

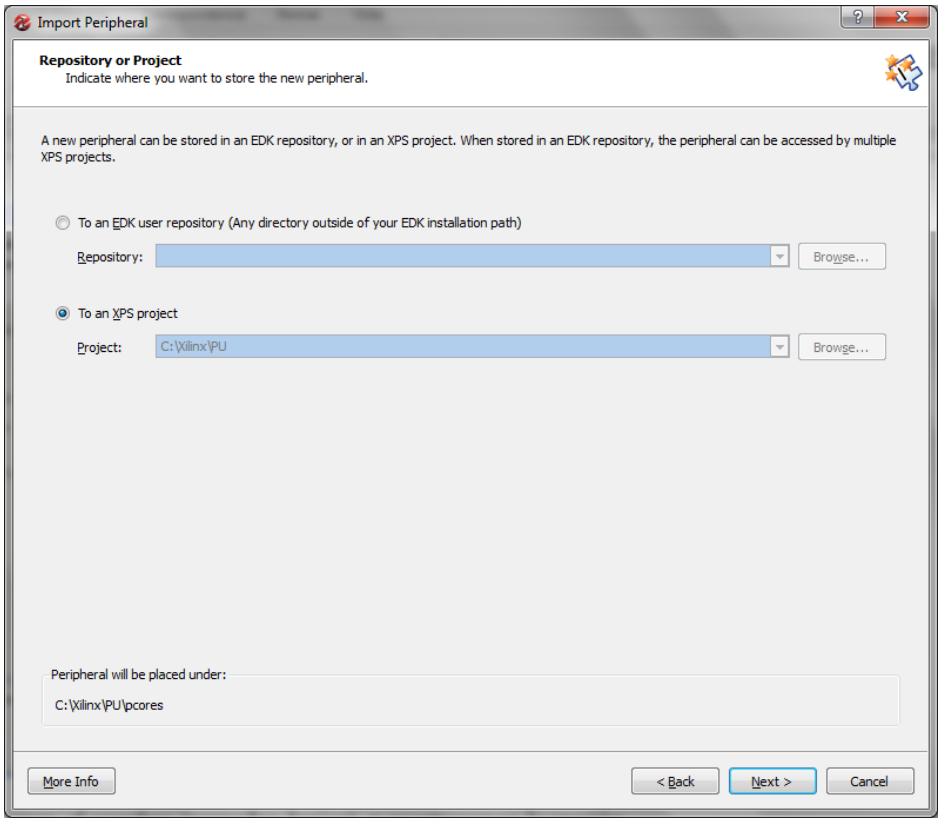
```
222
223     adder_PROC : process(slv_reg0, slv_reg1) is
224     begin
225         c(32) <= '0';
226         for i in 31 downto 0 loop
227         sum(i)  <= slv_reg0(i) xor slv_reg1(i) xor c(i+1);
228         c(i) <= (slv_reg0(i) and slv_reg1(i)) or (slv_reg0(i) and c(i+1)) or (slv_reg1(i) and c(i+1));
229         end loop;
230     end process adder_PROC;
231
```

Dejando el resto del código como está. No obstante, podemos analizar el código para entender como funciona el interfaz con nuestro periférico. Lo guardamos y cerramos.

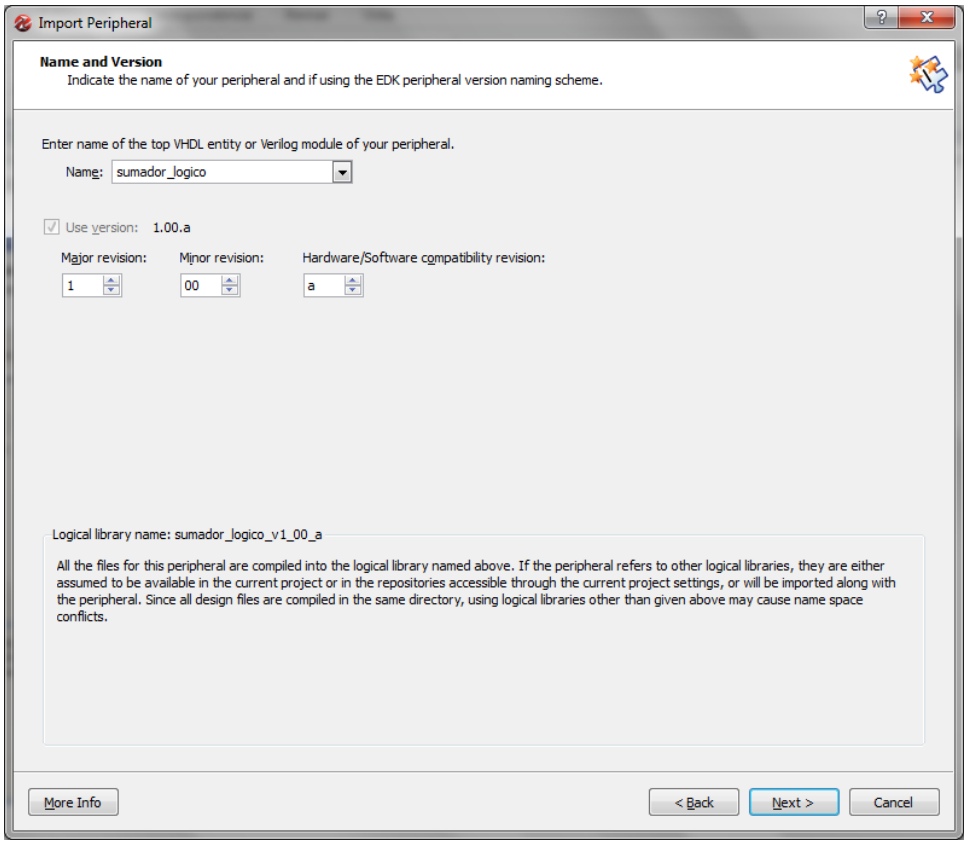
Ahora debemos importar el periférico creado a nuestro proyecto en EDK (Hardware -> Create or Import Peripheral). En este caso elegiremos la opción Import existing peripheral.



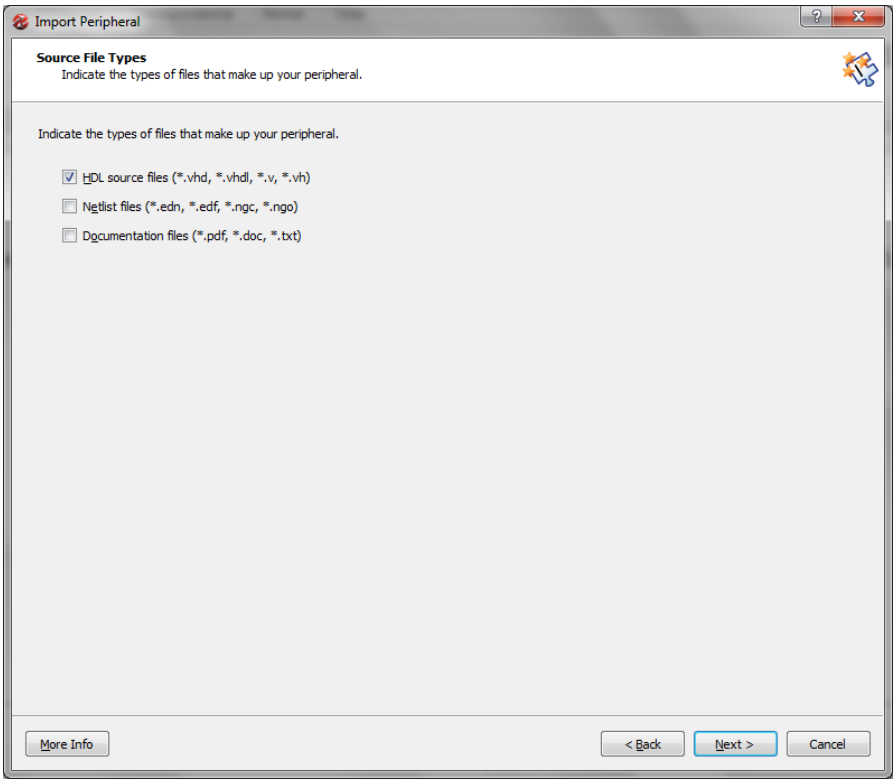
Seleccionamos donde queremos importarlo.

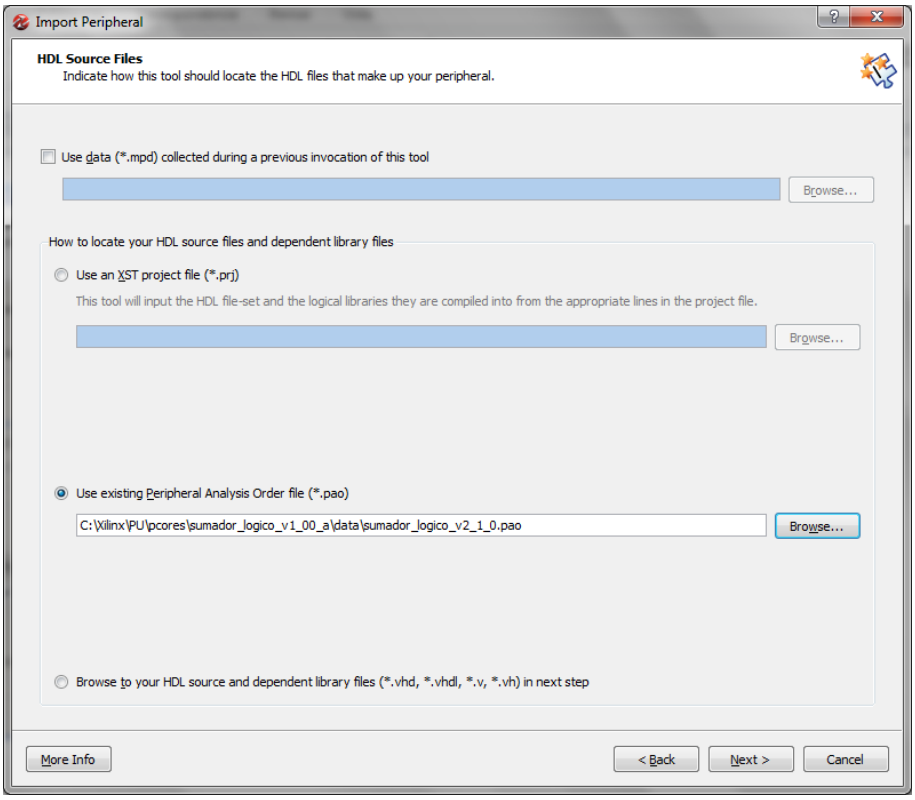


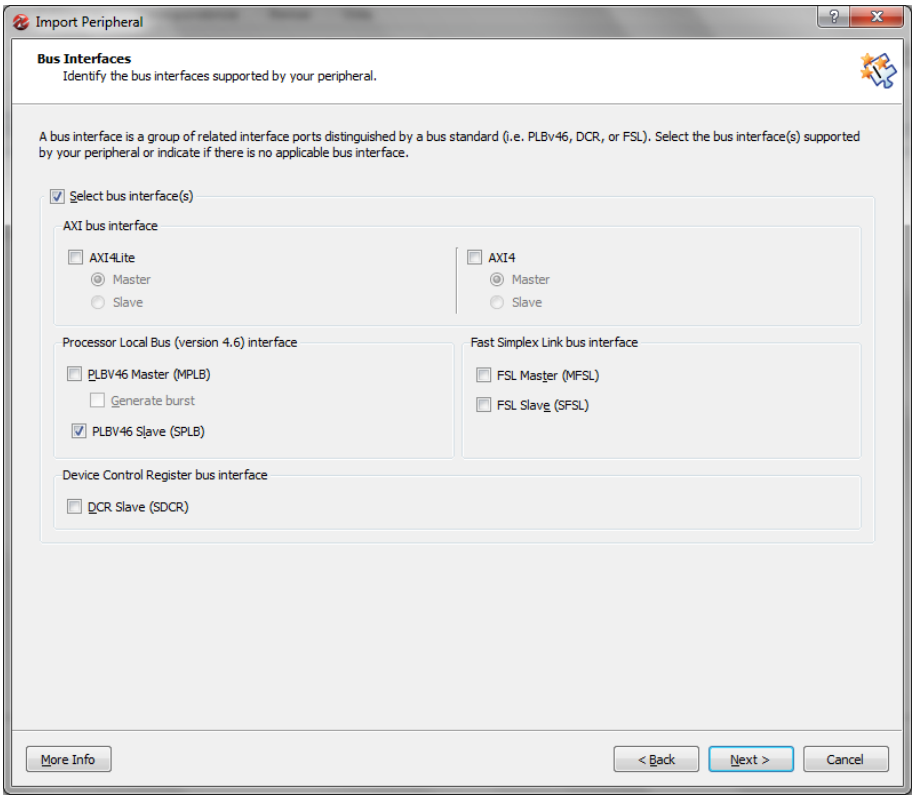
Seleccionamos el nombre (sumador_logico) y mantenemos la versión.



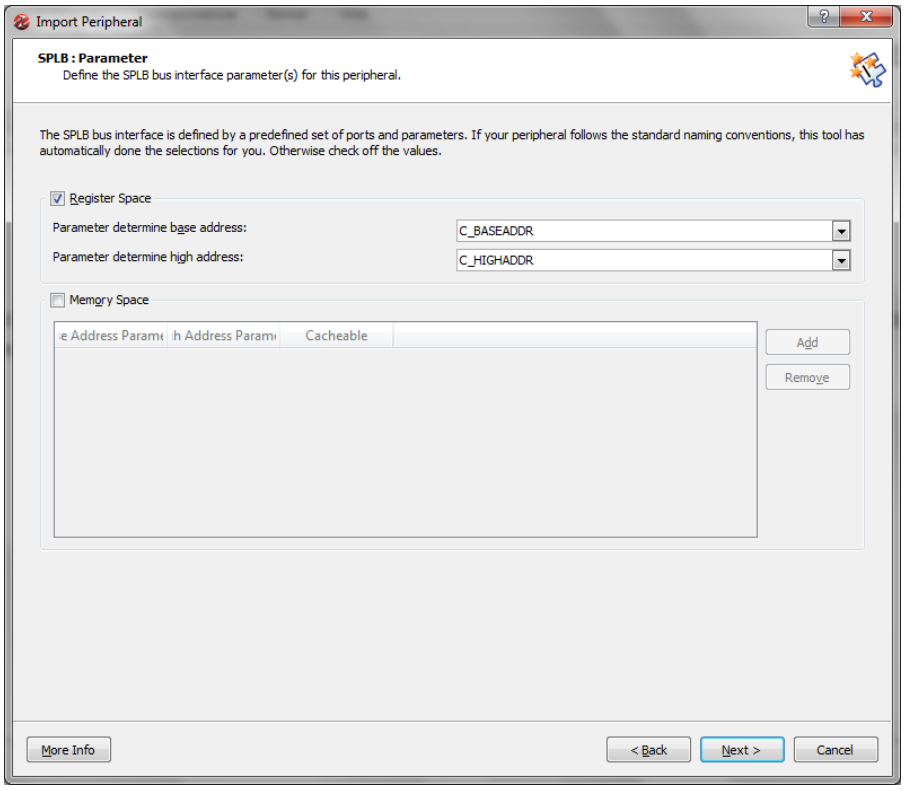
Le decimos que sí, para que permita la modificación de los ficheros del periférico y seleccionamos el tipo de archivos con el que hemos realizado nuestro periférico (HDL). Tomamos los archivos que ya han sido creados al generar el periférico y aceptamos todos los archivos que aparecen.



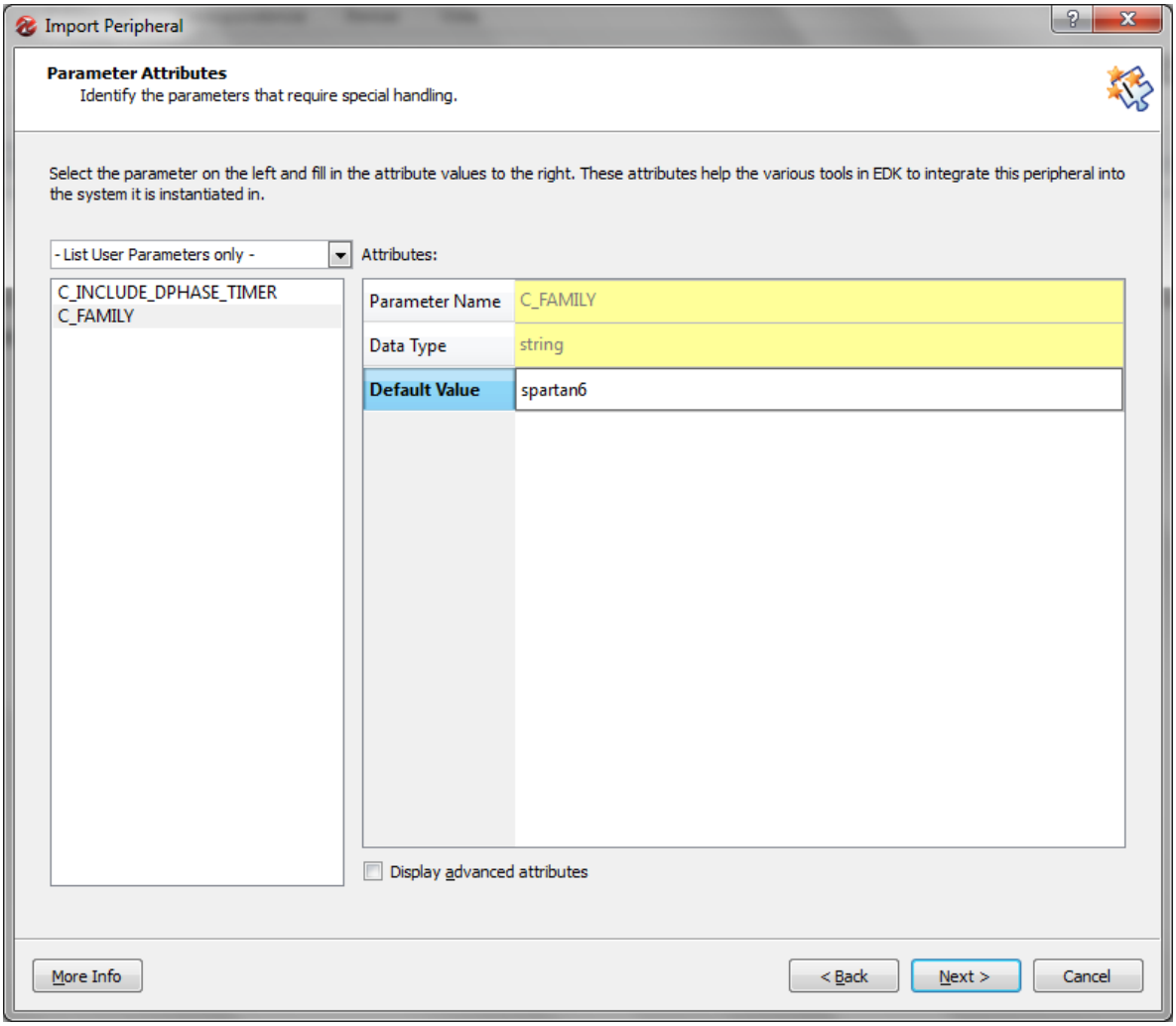




Actualizamos el espacio de registros tal y como muestra la siguiente imagen:



Aceptamos el interfaz de bus del periférico (el programa ha seleccionado correctamente cual es) y aceptamos las señales de interfaz del periférico. Por último, corregimos el valor de la familia de FPGA con la que estamos trabajando.



Finalizamos la importación y añadimos el periférico a nuestro proyecto como habitualmente hacemos. Lo conectamos al bus de microprocesador y generamos las direcciones. Al no haber salida al exterior, no hay que modificar nada de la pestaña Ports.

Ya estamos en disposición de usarlo y de comprobar su funcionamiento. Generamos el bitstream y, en caso de no encontrar errores, exportamos el hardware al SDK. En el SDK creamos el BSP y también una aplicación de test de periféricos, para comprobar que todo funciona correctamente.

Para poder usar éste nuevo periférico creamos una nueva aplicación y utilizamos el código que se muestra a continuación:

```
/************************** Function Definitions ***************************/
#define XPAR_SUMADOR_LOGICO_0_DEVICE_ID 0

#include "sumador_logico.h"
#include "xparameters.h"
#include "stdio.h"
#include "xstatus.h"
#include "sumador_logico.h"

int main (void) {

 Xuint32  *customLogicPtr;
 int a = 0xFFFFFFFF, b = 0x00000001, c = 0x12345678;

// Escribe en el puerto serie

 print("-- Entrando en el main() --\r\n\n");
 print("Software\r\n");
 c=a+b;
 xil_printf("%d + %d = %d \r\n", a,b,c);
 xil_printf("%x + %x = %x \r\n\n", a,b,c);

 customLogicPtr = (Xuint32 *) XPAR_SUMADOR_LOGICO_0_DEVICE_ID;


  // Escribe los valores en los registros 0 y 1

    *(customLogicPtr) = a;
    *(customLogicPtr + 0x1) = b;
    c = a;

    // Lee los valores del registro de la suma

    c = *(customLogicPtr + 0x2);

    print("Custom Logic\r\n");

    xil_printf("%d + %d = %d \r\n", a,b,c);
    xil_printf("%x + %x = %x \r\n\n", a,b,c);

while(1);

    print("-- Saliendo del main() --\r\n\n\n");

    return 0;

}
```

Comprobar su funcionamiento y realizar las modificaciones oportunas para que de los valores correctos. Comprender el funcionamiento basado en registros, y compararlo con el funcionamiento basado en FIFOs. ¿Cuando se realiza la suma hardware? ¿Funciona correctamente el código si eliminamos la línea "c=a;"? ¿Por que?

Ejercicio

9

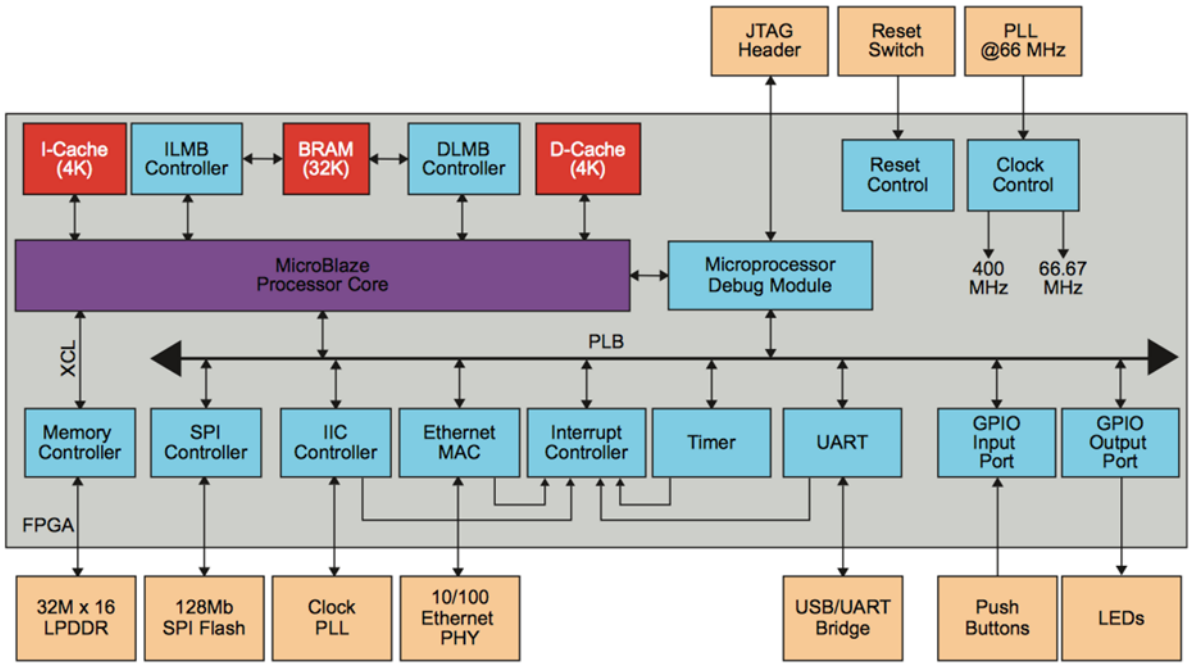
Codepuración hardware/software con SDK y ChipScope.

Usando ChipScope, realizaremos en este ejercicio una codepuración hardware/software que nos permita observar como evolucionan las señales dentro de la FPGA.

Usando el diseño hardware del Ejercicio_8, vamos a modificarlo para poder realizar en él la codepuración hardware/software que nos permitirá observar completamente nuestra aplicación. Para ello, usaremos ChipScope. Es un analizador lógico virtual para depuración hardware. Conectaremos y configuraremos los IPs ICON (Integrated Controller) e IBA (Integrated Bus Analyzer) de ChipScope que nos permitirán debugar el hardware diseñado.

A su vez, depuraremos la aplicación software desde SDK y usaremos ChipScope para visualizar las señales del bus PLB.

Este es el diagrama de bloques de la aplicación.

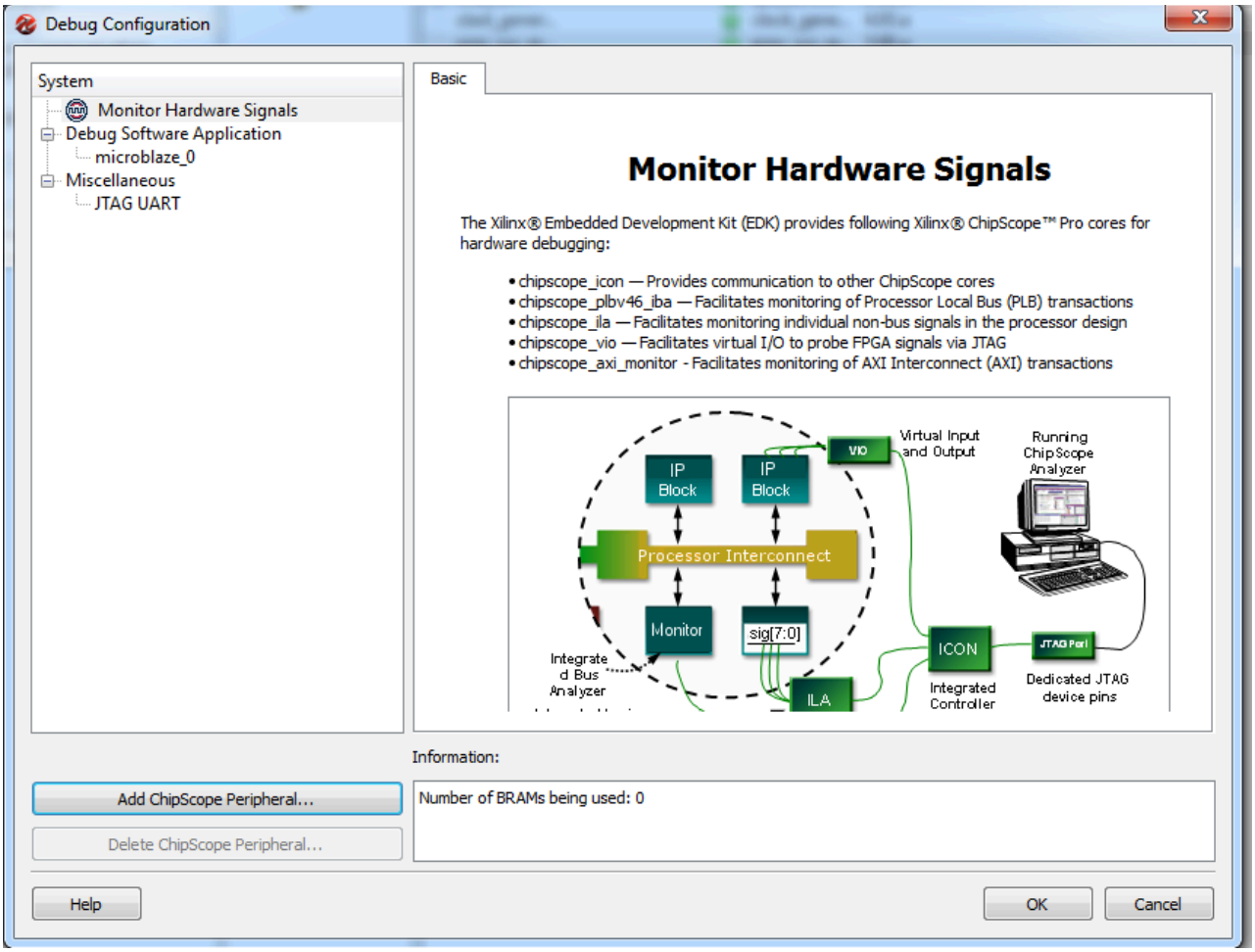


Copiamos el contenido del Ejercicio_8 dentro de una nueva carpeta (Ejercicio_9). Iniciamos EDK, abrimos el proyecto y generamos el bitstream para comprobar que antes de introducir ChipScope todo es correcto.

Observar los recursos ocupados por el sistema (Design Summary -> system) antes de introducir los cores de ChipScope (Flip Flops Usados, LUTs usadas y BRAMs usadas). Apuntarlo para luego compararlo cuando generemos el bitstream que incluye ChipScope.

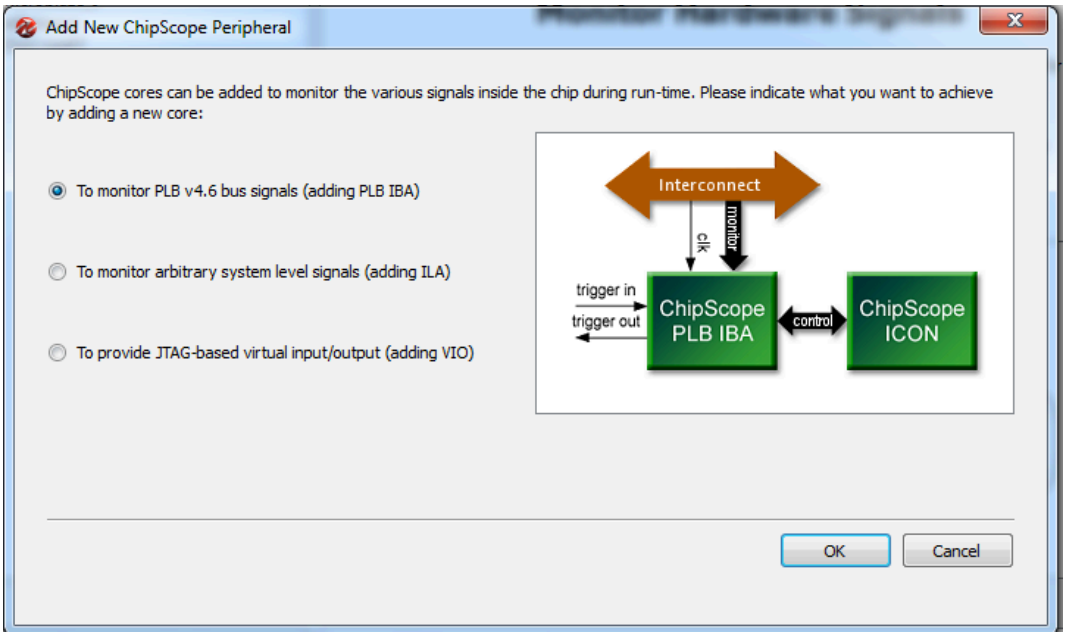
Para poder depurar con el analizador lógico integrado (ChipScope) debemos añadir, desde EDK, a nuestro sistema empotrado los periféricos chipscope_icon (Integrated Controller) y chipscope_plb_iba (Integrated Bus Analyzer).

Seleccionamos Debug -> Debug Configuration.

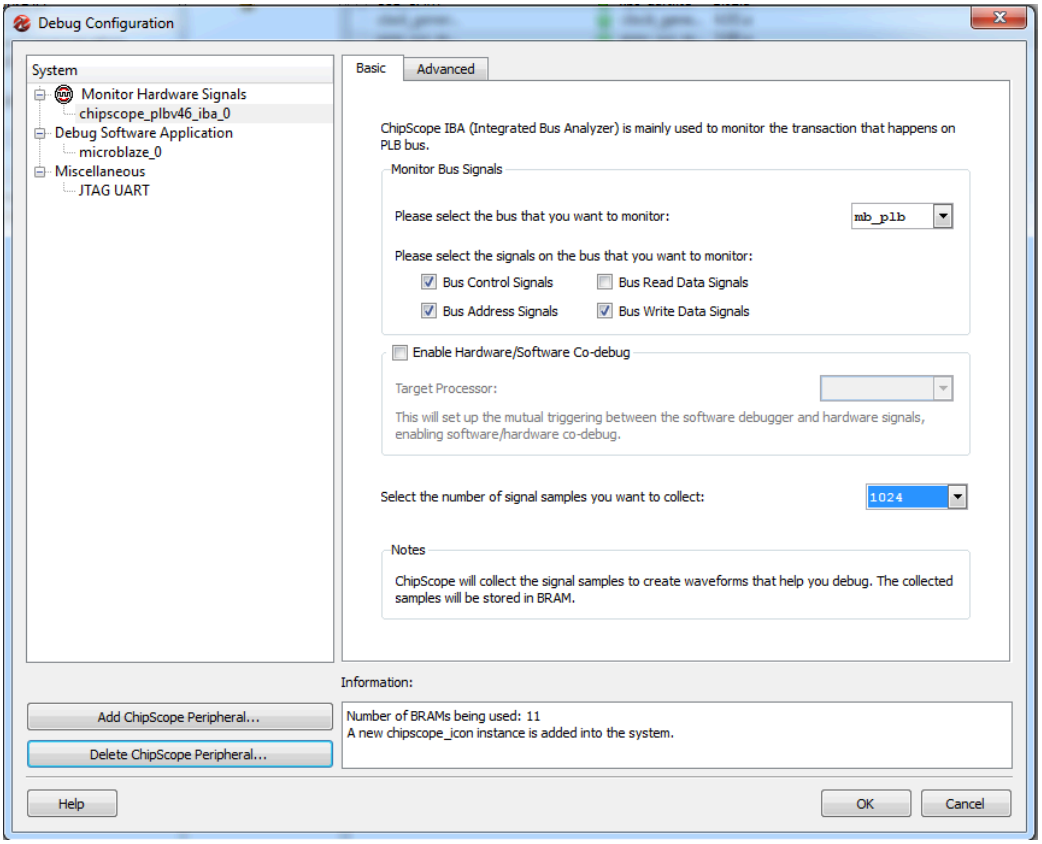


En esta pantalla podemos ver como se realiza la conexión entre los diferentes componentes para hacer una codepuración.

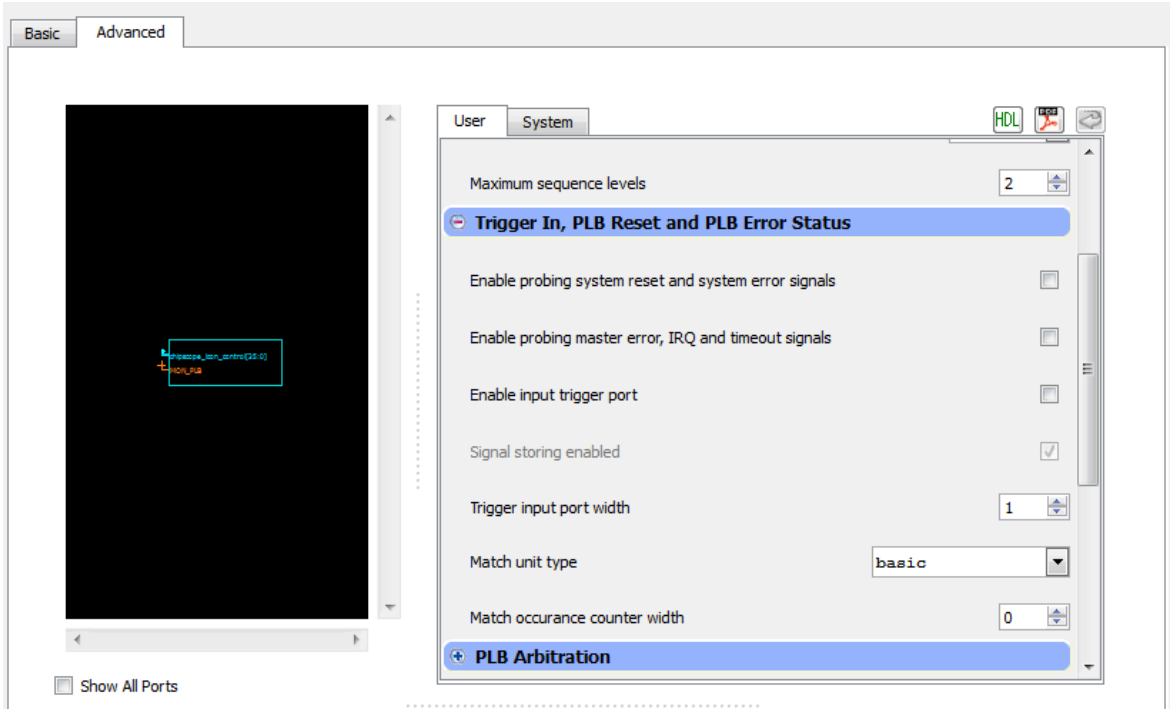
Seleccionamos Add ChipScope Peripheral y seleccionamos la primera opción (To monitor PLB v4.6 bus signals…).



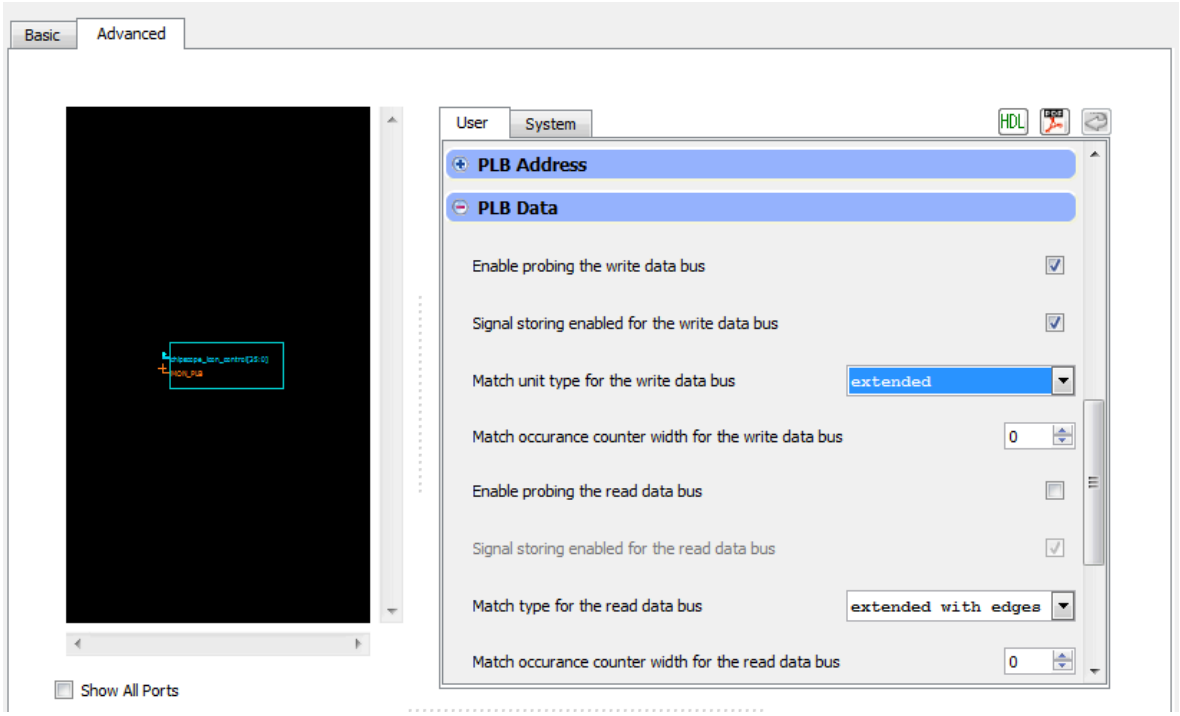
Marcamos el campo Bus Write Data Signals (así podremos comprobar que lo que escribimos en el bus es correcto) y dejamos el valor de Select the Number of signal samples you want to collect en 1024. Podríamos tomar muchas más muestras, pero eso significa usar más bloques BRAM.



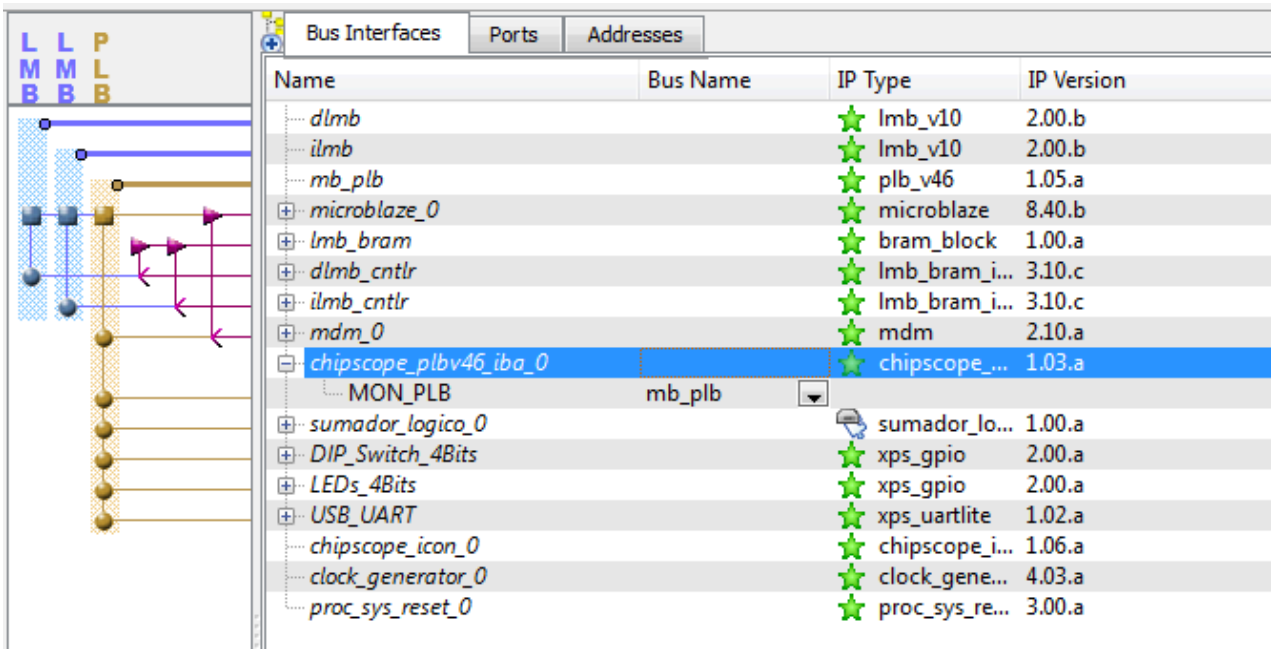
Pinchamos en la pestaña Advanced. Buscamos en la pestaña User el menú Trigger In, PLB Reset and PLB Error Status, deseleccionamos la opción Enable probing system reset and system error signals (así nos olvidamos de pruebas de reset) y cambiamos el valor de Match unit type a Basic (usamos los tipos de unidad básicos)



Seleccionamos ahora en el menú PLB Data la opción Match Unit Type for the write data bus y cambiamos su valor por extended. Nos permitirá trabajar con comparaciones.



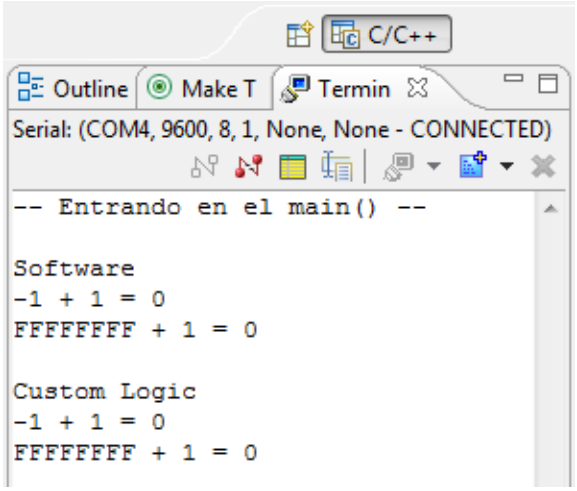
El resto de parámetros se dejan con los valores por defecto. Observamos el sistema empotrado y vemos como se han añadido los dos periféricos que nos permitirán realizar la codepuración.



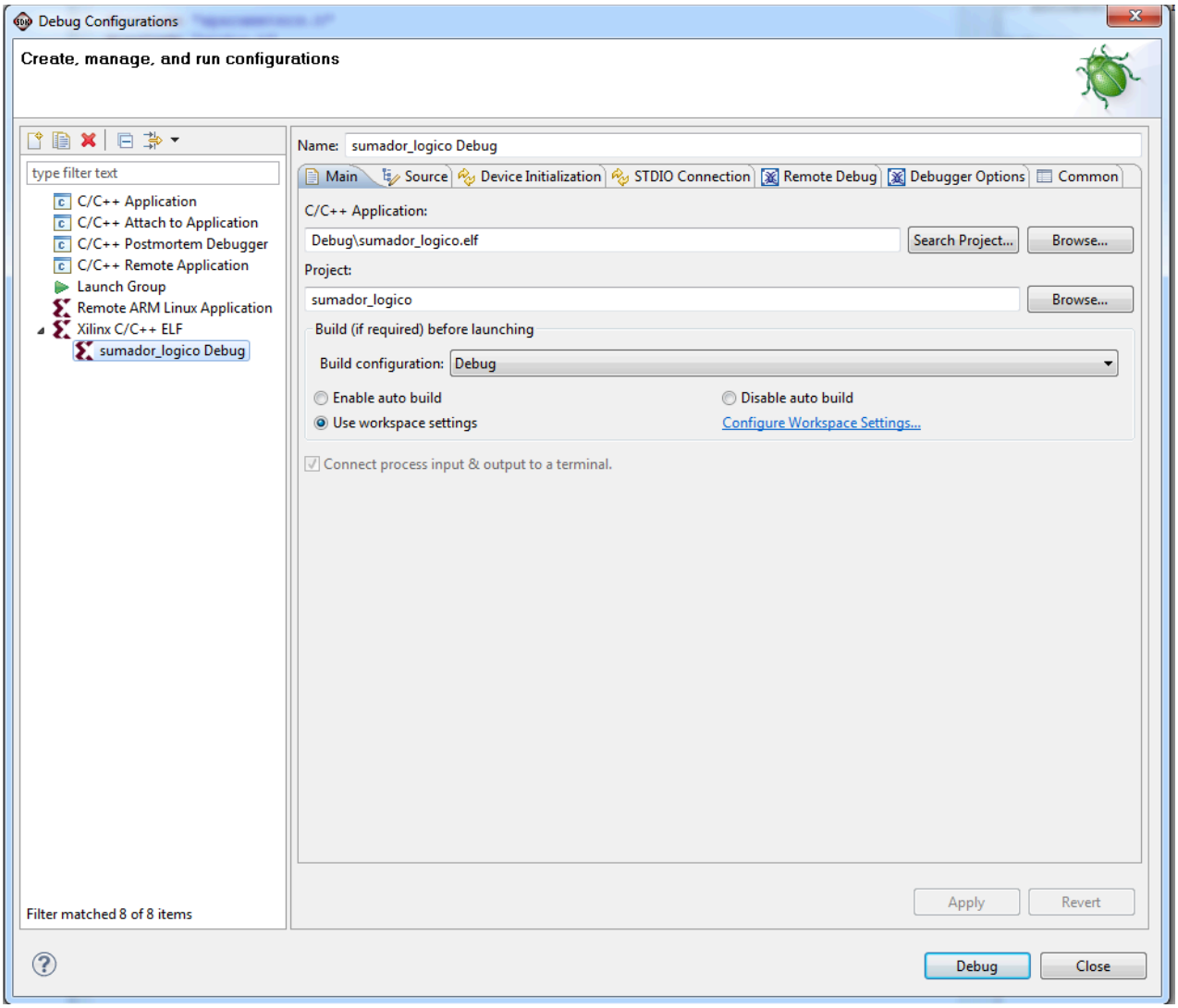
Ahora podemos implementar el nuevo hardware creado para obtener el archivo de programación (Generate Bitstream). Cuando termine, observar los recursos ocupados y compararlos con los apuntados al inicio del Ejercicio.

Como queremos depurar nuestro hardware mientras corremos el software, abrimos SDK y mantenemos la aplicación creada en el Ejercicio_8.

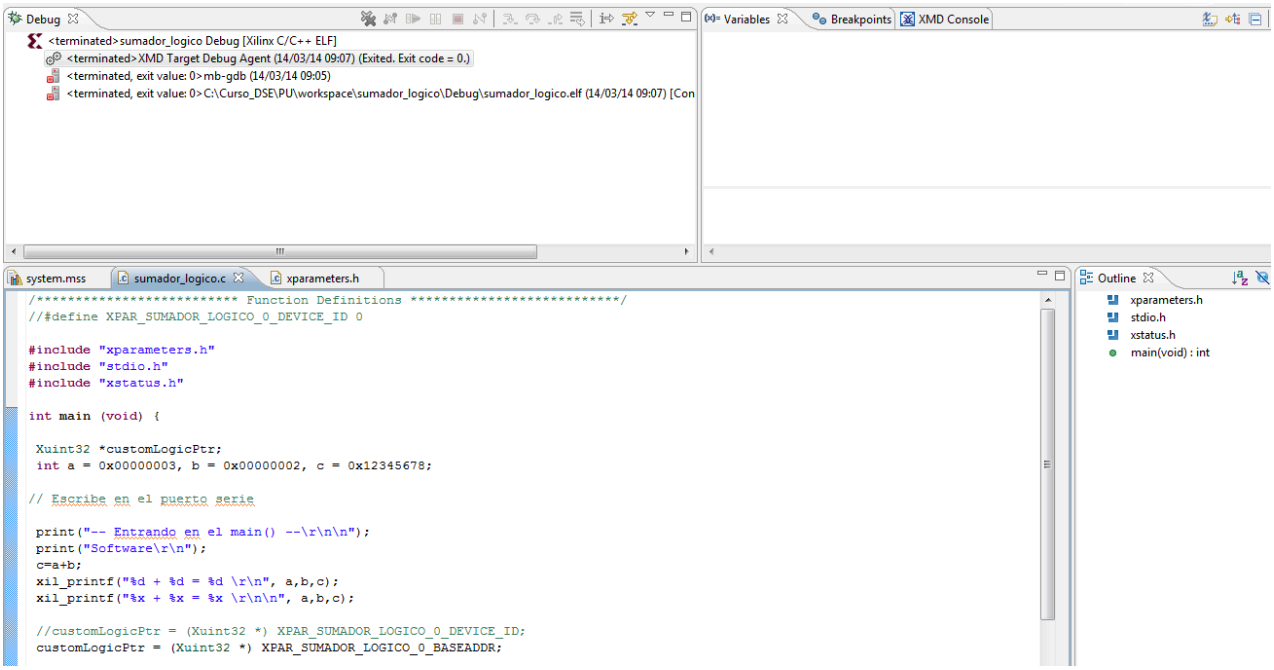
Para iniciar la depuración, programamos la FPGA desde SDK. Comprobamos por Terminal que la suma se realiza correctamente.



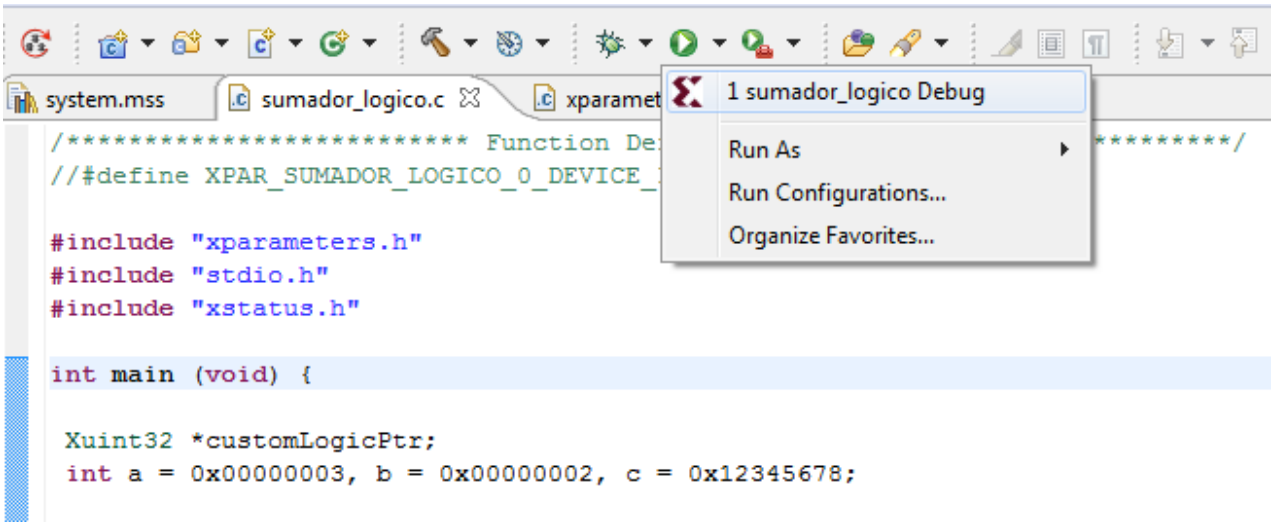
Ejecutamos la opción Run -> Debug Configurations -> Xilinx C/C++ ELF creando una nueva configuración para la depuración del Sumador Lógico.



Entraremos en la ventana de depuración del entorno SDK, conectado directamente con la FPGA. El programa se queda en espera en el main hasta ser ejecutado.

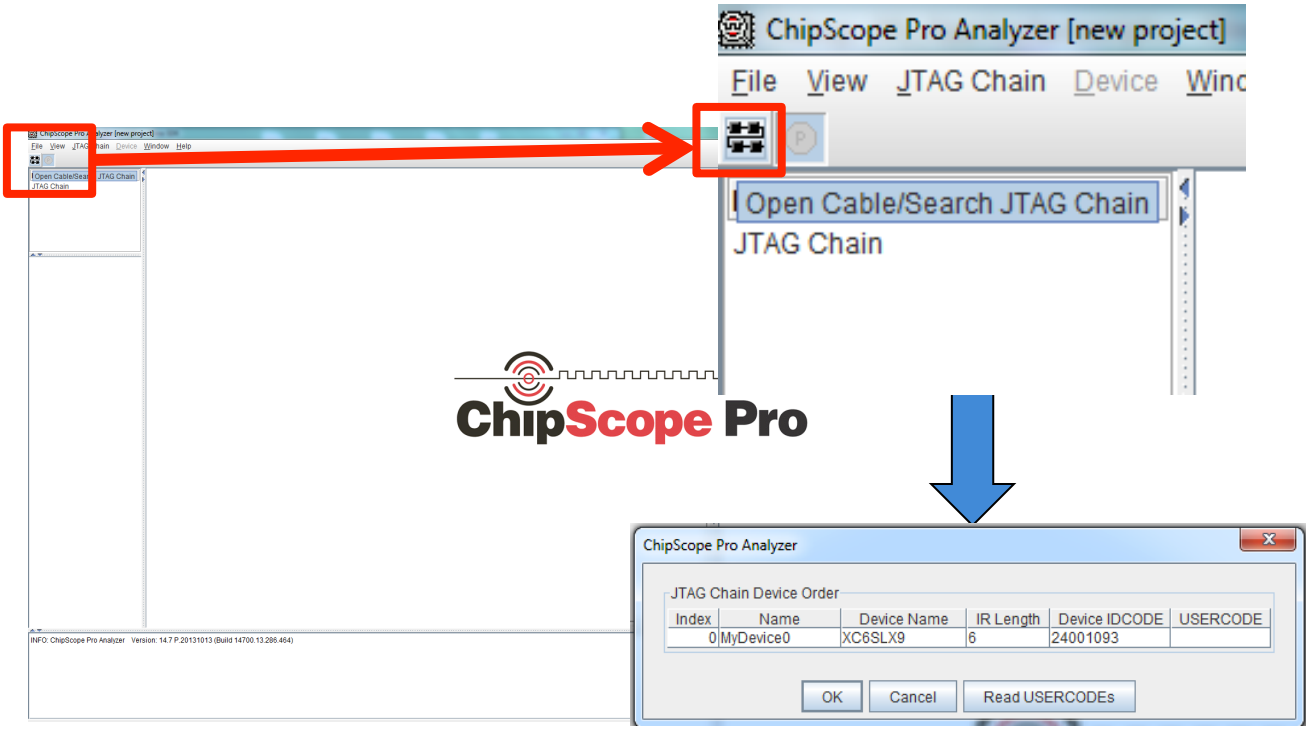


Si no funciona, podemos correr directamente el software en la FPGA para probar que ChipScope sí que analiza las señales internas de nuestro sistema. Para ello, clicar sobre el icono verde de Run y ahí seleccionar sumador_logico_debug.

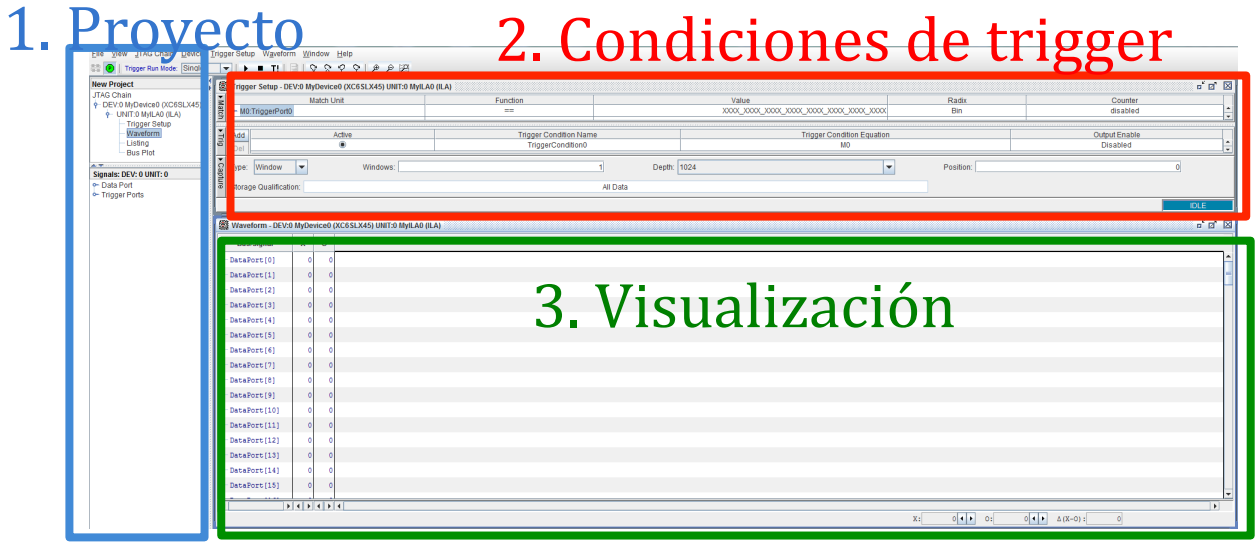


Para realizar la depuración hardware debemos abrir el programa analizador de señales Chipscope Pro Analyzer. Buscarlo dentro de ISE Design Suite 14.4.

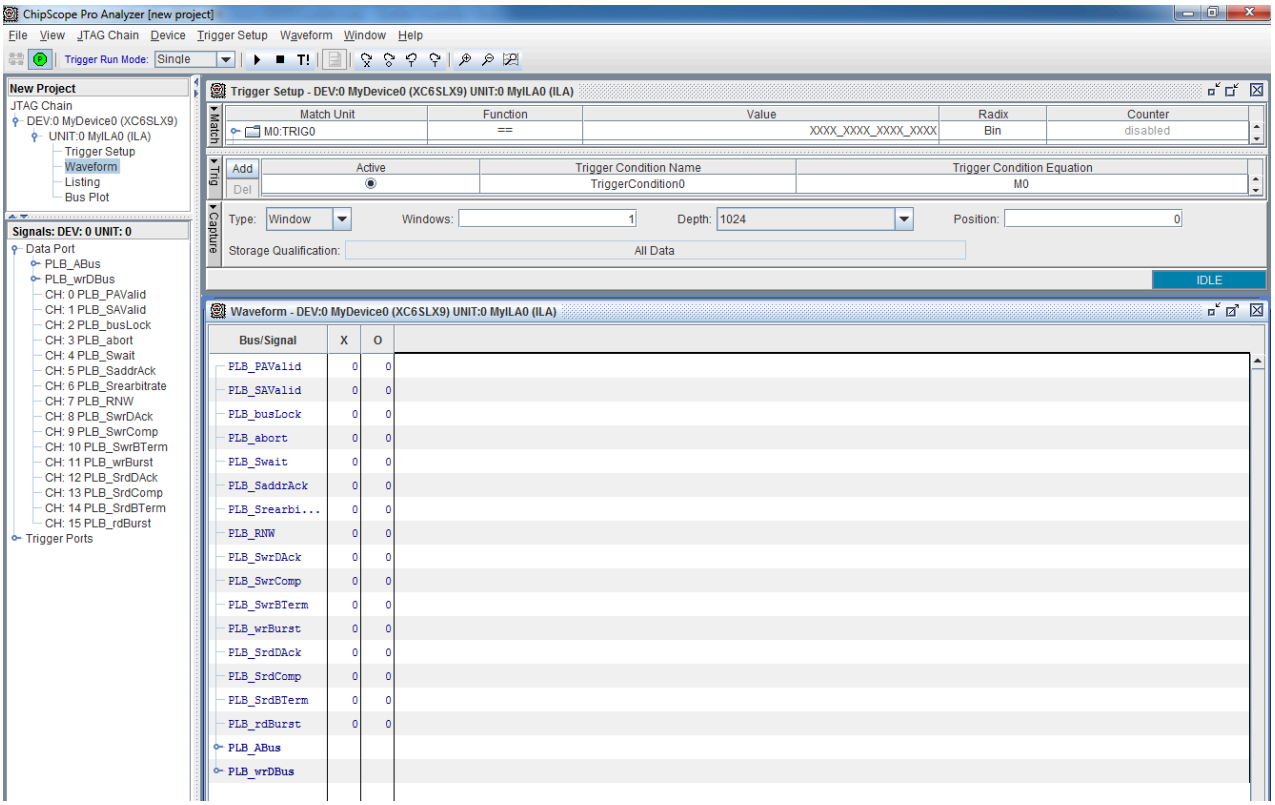
Con la Opción Open Cable/Search JTAG Chain, nos conectaremos a la FPGA y detectaremos la Spartan 6 con la cual estamos trabajando.



El programa Chipscope identifica los dispositivos conectados a la cadena de programación (Boundary Scan). Si son los correctos, continuamos para realizar la simulación. La estructura de ChipScope es la siguiente.

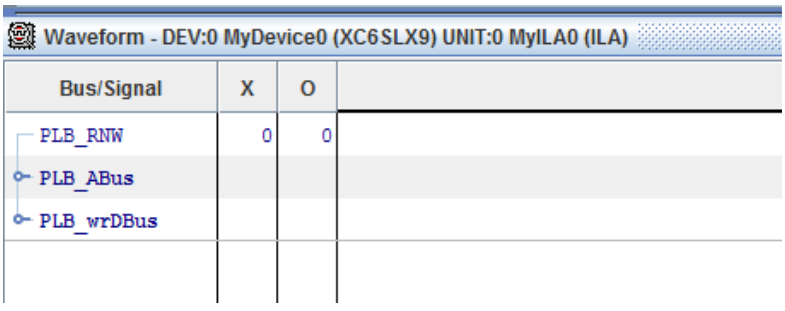


Una vez conectado a la FPGA nos aparecen las ventanas de señales que estamos usando (Signals), las ventanas de disparo (Trigger) y las ventanas con las formas de onda (Waveform).



Seleccionamos la opción File -> Import y el botón Select New File. Buscamos el archivo chipscope_plbv46_iba_0.cdc que está en el directorio de nuestro ejercicio en la carpeta implementation\chipscope_plb46_iba_0_wrapper. Obtenemos todas las señales que usamos en este ejercicio y que podemos observar.

Borramos todas las señales de la ventana Waveform (Remove From Viewer) excepto la señal PLB_RNW y las señales PLB_Abus y PLB_WrDBus.



Como en cualquier Analizador Lógico, antes de nada, debemos ajustar las condiciones de disparo en la ventana Trigger.

En la Ventana Match, debéis entender la señal PLB_RNW (en la unidad M0:TRIG0), es con la que tenemos que definir nuestra primera señal de disparo o trigger. Darle el valor adecuado, lo más cómodo en este caso es que el valor de comparación esté en Binario.

Cambiamos el código de comparación de la unidad M1:TRIG1 de binario a hexadecimal. Pensar en este caso qué se define en dicha unidad y poner el valor correcto.

Cambiamos el código de comparación de la unidad M2:TRIG2 de binario a Unsigned. Igualmente, pensar qué estamos definiendo y que valor debemos colocar.

En la Ventana Trigger, seleccionamos la condición de disparo pinchando la opción Trigger Condition Equation de la condición de disparo TriggerCondition0.

La condición de disparo que nos interesa es la combinación de M0, M1 y M2, pensar cuando queremos que salte el Trigger teniendo en cuenta lo que hace el Sumador_Lógico creado.

En la Ventana Capture, cambiamos el valor de la posición de disparo a 0, en la opción Position, para que nos muestre los valores anteriores y posteriores del disparo (tomamos 32 muestras, Depth)

Así mismo, definimos la condición para la captura de valores del Bus (Storage Qualification) jugando con M0, M1 y M2.

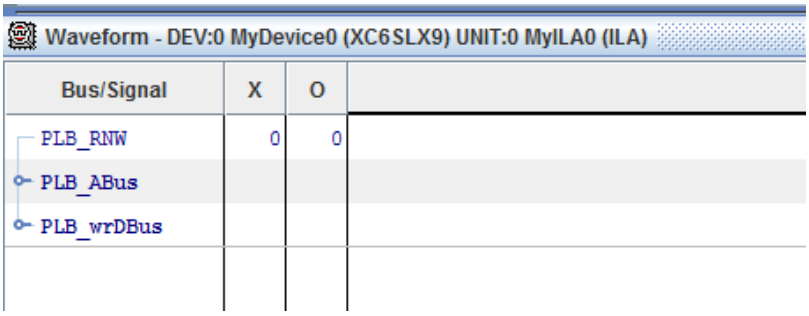
Con ello debemos conseguir que cuando el valor enviado por MicroBlaze hacia los registros del Sumador_Lógico sea el que nosotros hemos puesto en SDK salte el trigger y nos muestre exactamente lo que circula por el bus de direcciones y de datos.

También deberíamos ver qué ocurre cuando se produce el resultado de la suma y este valor se envía hacia MicroBlaze.

Grabamos el proyecto de configuración de Chipscope en el directorio del ejercicio e iniciamos la adquisición mediante la opción Trigger Setup -> Run.

El analizador se queda esperando a que se produzca la condición de disparo que le hemos marcado.

Como estamos haciendo codepuración hardware/software, desde SDK haremos que corra el programa del microprocesador empotrado en la FPGA (Run).



Deberíamos ver como la señal PLB_RNW se pone a cero o a uno y como el disparo se ha producido cuando se cumple la condición marcada por M1:TRIG1 y M2:TRIG2.

Se os propone la colocación de puntos de ruptura en SDK que os permitan ejecutar el programa mientras el analizador Chipscope está activo. Así podréis controlar la ejecución del software y ver las señales hardware en el momento del disparo.

Asimismo, también se os propone trabajar con valores de suma más sencillos al principio para comprobar si funciona correctamente lo definido en ChipScope.

Recordar que esta simulación es totalmente hardware, estamos observando que ocurre en el interior de la FPGA. Los resultados obtenidos son totalmente fiables.

Ejercicio

10

Control mediante SPI de un Termómetro Digital.

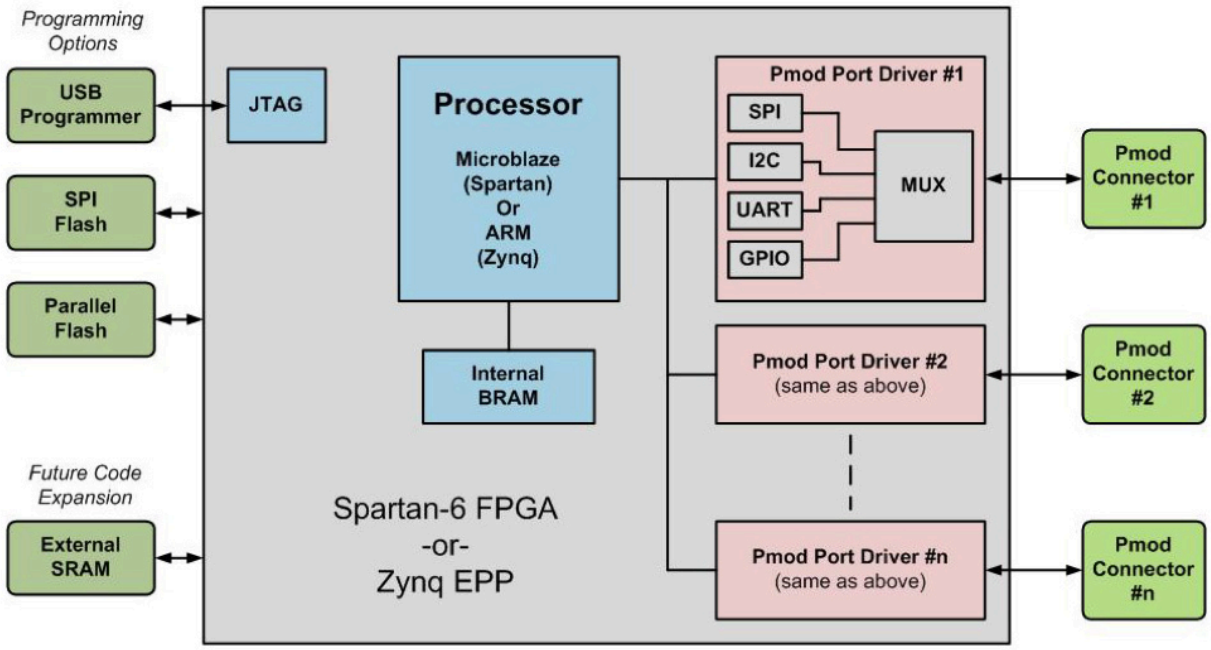
Utilizando el Pmod MAX31723, realizaremos en este ejercicio el control del mismo que nos permita observar la temperatura ambiente conectándonos vía SPI con la FPGA.

A partir de un Proyecto base que nos ofrece Maxim, abriremos SDK para cargar el hardware, las librerías necesarias y los diferentes archivos en C que nos permitan manejar el Pmod MAX31723. Dicho proyecto está preparado no sólo para trabajar con este Periférico si no con todos los que se incluyen en el *"Analog Essentials Collection"* de Maxim.

Aprenderemos a programar sólo el hardware en la memoria BRAM interna de la FPGA y el software en la DDR externa de la Spartan LX9. Esto es debido a que el software ocupa más memoria de la que tenemos internamente en la FPGA.

Veremos los menús que nos da Maxim para elegir con qué Pmod vamos a trabajar, elegiremos el Pmod de Temperatura y observaremos la misma. Así como definir alarmas en caso de superar ciertos márgenes.

Este es el diagrama de bloques del Proyecto que nos ofrece Maxim.



En la siguiente Tabla podemos ver todos los periféricos que nos ofrece Maxim en su *"Analog Essentials Collection"*.

Part	Functionality	Interface
DS1086LPMB1#	I^2C spread-spectrum EconOscillator™	I^2C
DS3231MPMB1#	I^2C real-time clock	I^2C
MAX3232PMB1#	RS-232 transceiver	UART
MAX4824PMB1#	Octal relay driver	GPIO
MAX5216PMB1#	High-performance 16-bit DAC	SPI/GPIO
MAX5487PMB1#	Dual 256-tap digital potentiometer	SPI
MAX5825PMB1#	Octal 12-bit DAC	I^2C
MAX7304PMB1#	16-port GPIO and LED driver	I^2C
MAX9611PMB1#	Current-sense amplifier with op amp and ADC	I^2C
MAX11205PMB1#	16-bit delta-sigma ADC with 2-wire interface	GPIO
MAX14840PMB1#	RS-485 transceiver	UART/GPIO
MAX14850PMB1#	6-channel, 600V galvanic isolator	SPI/UART
MAX31723PMB1#	Digital thermometer	SPI
MAX31855PMB1#	Thermocouple-to-digital converter	SPI
MAX44000PMB1#	Ambient light and proximity sensor	I^2C

Podemos observar, además de para qué sirve cada uno, el interfaz de comunicación que necesitamos manejar para poder trabajar con ellos.

En nuestro caso, MAX31723 Digital Thermometer, necesitamos funcionar con SPI.

Uso del Proyecto de Maxim desde SDK

En pequeños diseños hechos con Microblaze en FPGAs, como ha sido hasta ahora, es típico que el programa ejecutable quepa totalmente dentro de la RAM interna de la misma. En estos casos, la FPGA se puede programar simplemente descargando un archivo único, que contiene tanto el hardware definido como el ejecutable para el software que se ejecuta en el procesador.

Sin embargo, el tamaño del programa ejemplo para la colección de módulos Pmods analógicos de Maxim excede la capacidad de la memoria RAM interna. Por lo tanto, la memoria externa DDR (fuera de la FPGA) debe ser utilizada tanto para el programa como para el almacenamiento de datos. Debido a esto, la carga del programa es un proceso de múltiples pasos.

Para todo ello, vamos a seguir los siguientes pasos:

1. Descargar el v1_4.zip del Aula Virtual y descomprimirlo en el raíz del disco duro del ordenador.

2. Abrir Xilinx -> SDK.

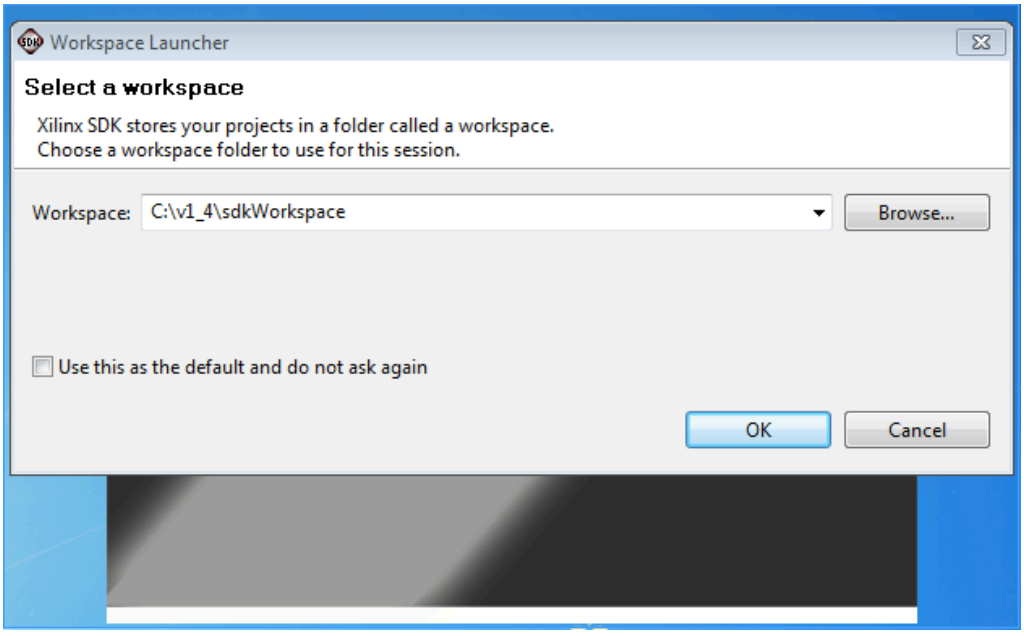
3. Descargar el bitstream (.bit) en la FPGA. Este bitstream contiene el diseño hardware con MicroBlaze.

4. Abrir Terminal para poder elegir el Pmod con el cual vamos a trabajar y después para observar los datos recibidos.

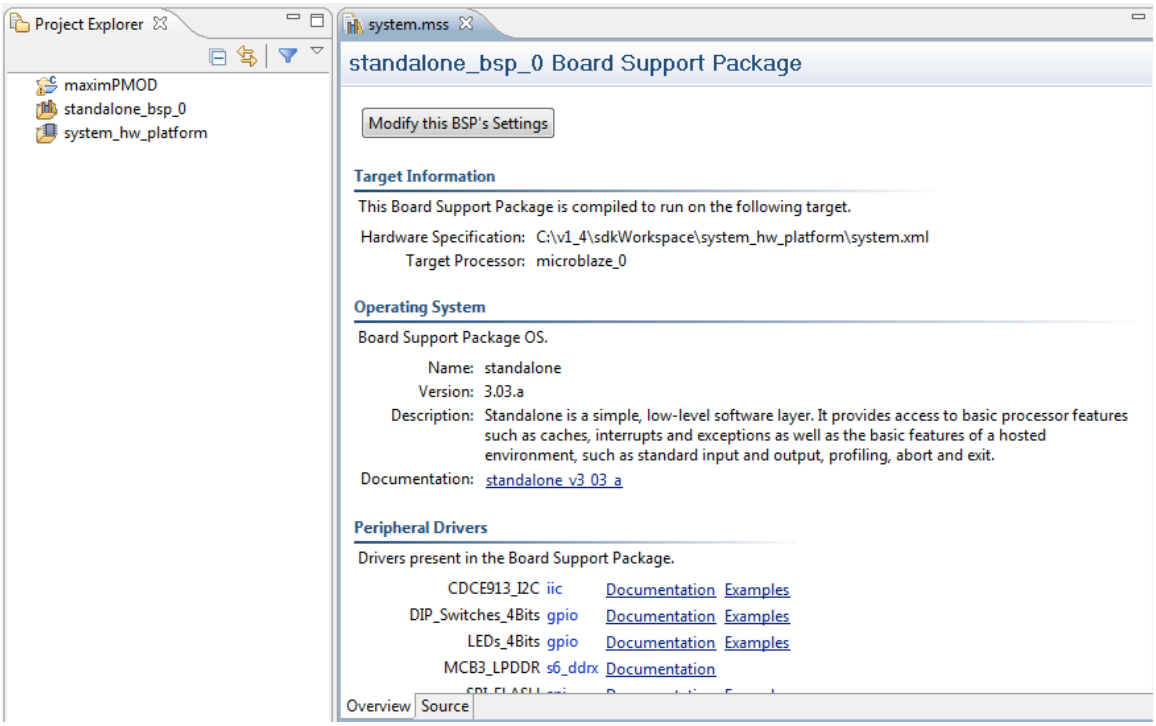
5. Usar la consola de depuración (Xilinx SDK/XMD) para descargar y correr el software (executable file .elf) en Microblaze desde la DDR externa.

Abrir Proyecto en SDK

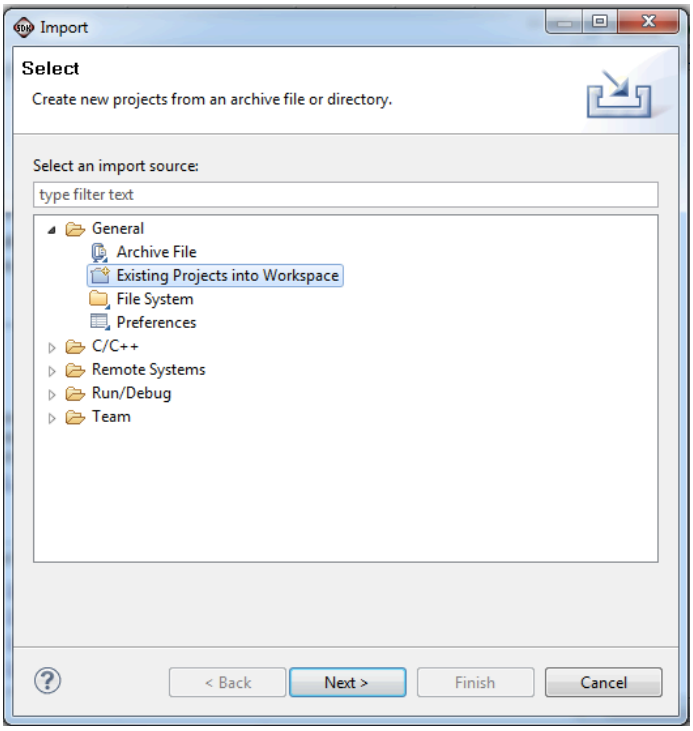
Abrimos SDK, cuando nos pida seleccionar el Workspace con el cual vamos a trabajar debemos buscar lo siguiente.



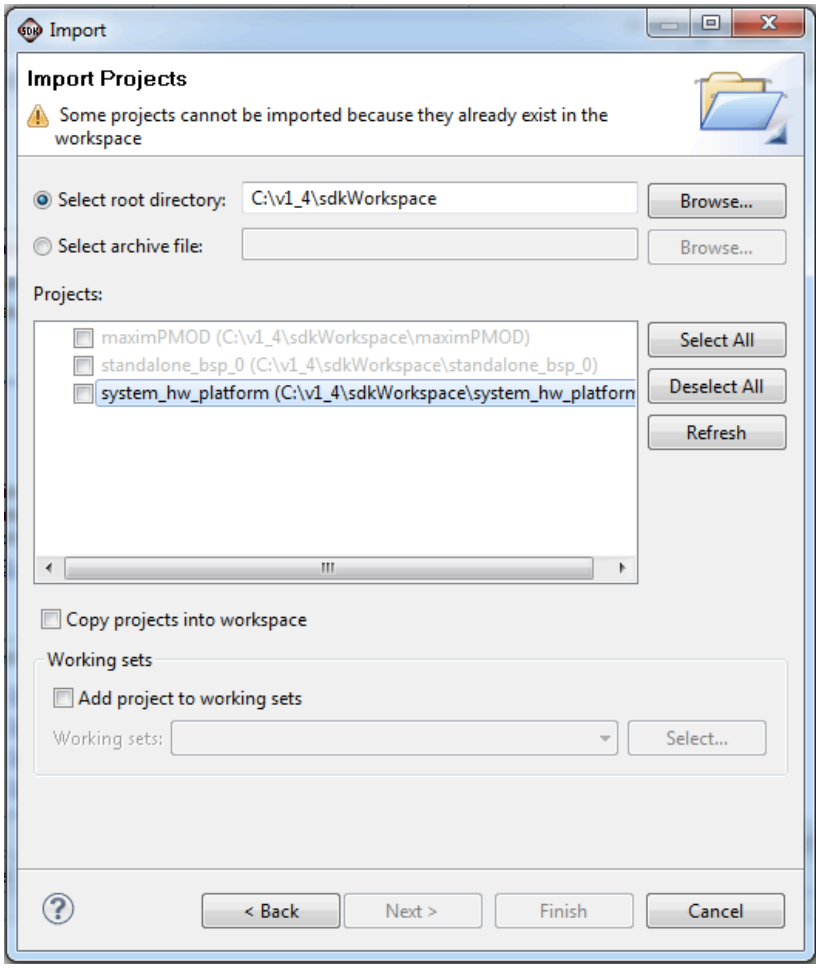
Revisar el SDK IDE. El Explorador de proyectos en la pestaña superior izquierda debe tener tres componentes como se muestra en la imagen de abajo.



Si las tres subcarpetas están presentes, podremos empezar a programar la FPGA. Si el Explorador de proyectos no contiene estas tres subcarpetas, abrir el menú File -> Import, expandir la ficha General y seleccionar Existing Projects into Workspace.



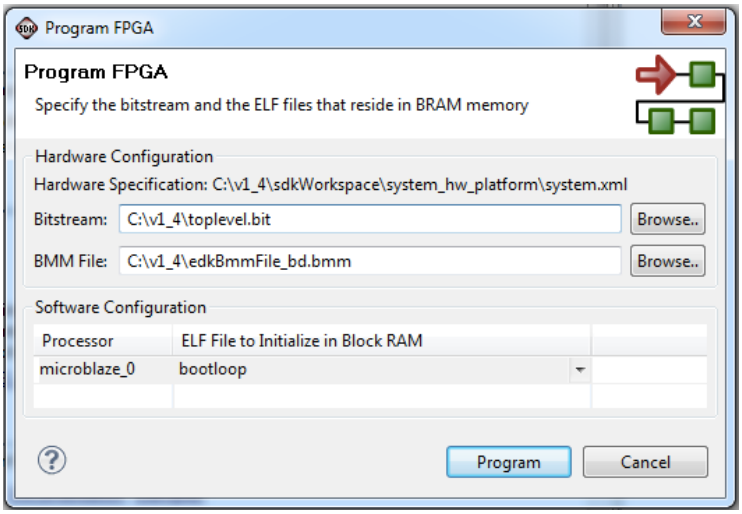
Hacer clic en Next y establecer el directorio raíz de C:\v1_4\sdkWorkspace, los proyectos que faltan deben aparecer en el explorador SDK de Project con sus casillas marcadas. Hacer clic en Finish para importar los proyectos.



Descarga del hardware en la FPGA

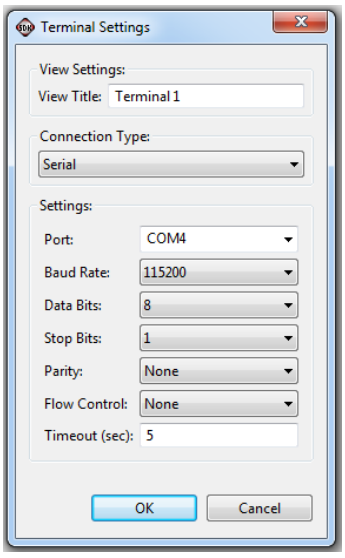
En este paso vamos a descargar el bitstream (.bit) en la FPGA. Para ello le damos al icono Program FPGA. Aparecerá el cuadro de diálogo para programar la FPGA. A partir de aquí, seleccionamos el .bit que contiene el hardware que queremos integrar en la FPGA, el archivo de memoria .bmm y el bootloop para cargar en la memoria interna (BRAM) de la FPGA.

Aseguraos seleccionar el archivo toplevel.bit, el edkBmmFile_bd.bmm y el archivo bootloop como se muestra a continuación, seleccionar Program.



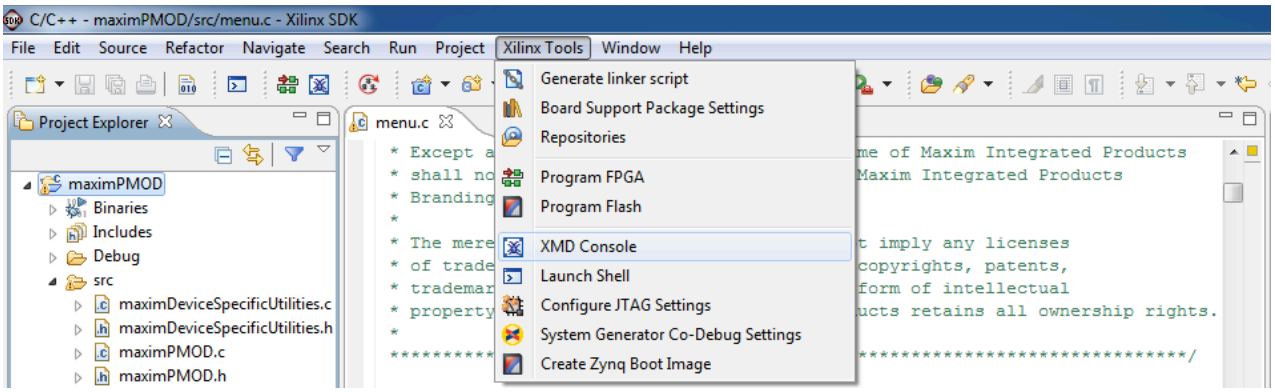
Como se comentó anteriormente, el tamaño del código de este software excede el disponible en la memoria interna de la FPGA, lo que exige el uso de la DDR externa. Esto requiere un proceso de carga de varias etapas implementado con un sistema de arranque y funciona un poco diferente a los programas más pequeños, donde toda la información de configuración de la FPGA y código ejecutable está contenida en un archivo de bits y se carga en un solo paso.

Antes de continuar y terminar de programar la FPGA, debemos conectar Terminal para poder manejar el Menú de Maxim y después observar los valores de Temperatura que nos da el Pmod.



Descarga del software en la DDR

Ahora es momento de descargar el software en la memoria DDR usando la aplicación Xilinx SDK/XMD. Para ello, hacer click en la pestaña XMD Console. Si no aparece, ir a Xilinx Tools y ahí lo tenéis.

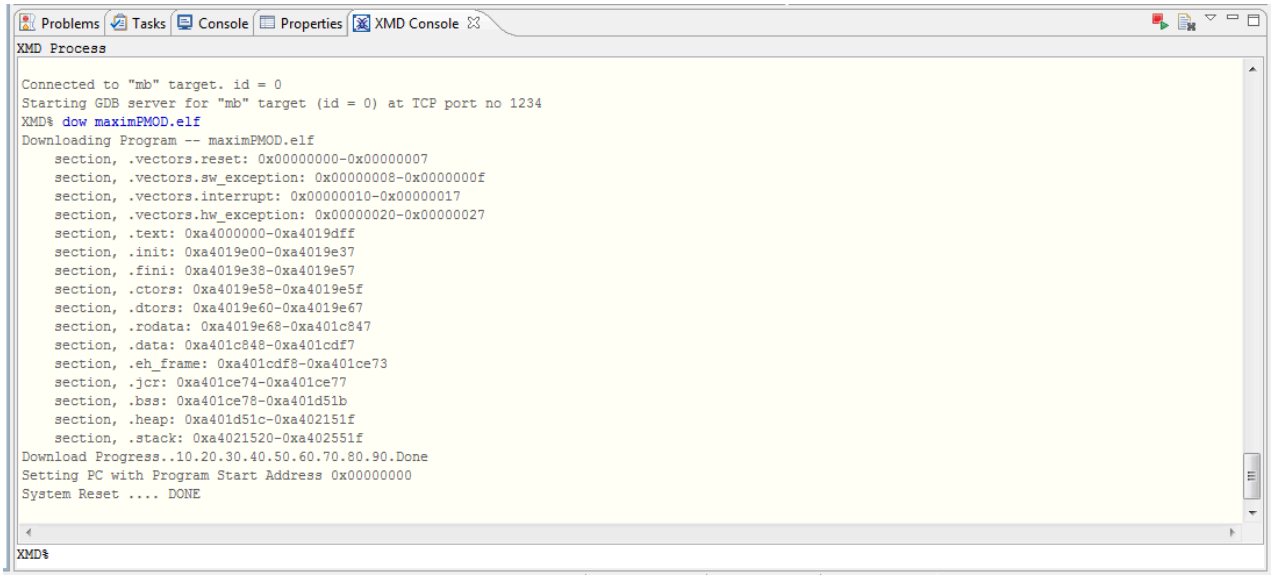


En el **XMD%** prompt en la zona inferior del XMD Console, realizar las siguientes acciones.

1. Usar "pwd", "cd" y "ls" para llegar al directorio -> **C:\v1_4\sdkWorkspace\maximPMOD\Debug**

2. Escribir lo siguiente para conectarnos con la memoria -> **connect mb mdm**

3. Descargar el archivo software mediante -> **dow maximPMOD.elf**



4. Esperar un tiempo hasta que termine la descarga en la DDR y entonces escribir -> **run**

En este punto, la aplicación se ejecuta en MicroBlaze y por USB UART/Terminal debe aparecer un menú como el de abajo. Este es el menú de nivel superior para los programas de ejemplo de los módulos Pmod de Maxim.

```
////////////////////////////////////////////////////////////////////
//                                                                //
//         ##     ##     ##     #  #  #######  ##     ##         //
//         # #   # #    #  #     ##      #     # #   # #         //
//         #  # #  #   ######    ##      #     #  # #  #         //
//         #   #   #  #      #  #  #  #######  #   #   #         //
//                                                                //
////////////////////////////////////////////////////////////////////

Press a key/number to test a Peripheral Module:
{0} DS1086L   Spread Spectrum Econ Oscillator
{1} DS3231M   I2C Real-Time Clock
{2} MAX3232   RS-232 Transceiver
{3} MAX4824   Octal Relay Driver
{4} MAX5216   SPI-Compatible, High performance 16 bit DAC
{5} MAX5487   Dual, 256-tap SPI Digital Potentiometer
{6} MAX5825   I2C Octal DAC
{7} MAX7304   I2C-Interfaced 16-Port, GPIO and LED Driver
{8} MAX9611   Current sense amplifier with OpAmp and ADC
{9} MAX11205  16 bit Delta-Sigma ADC with 2-Wire interface
{A} MAX14840  RS-485 Transceiver
{B} MAX31723  DS31723 Digital Thermometer
{C} MAX31855  Thermocouple to Digital Converter
{D} MAX44000  I2C-Interfaced Ambient Light and Proximity Sensor
```

Utilización del Pmod de Temperatura

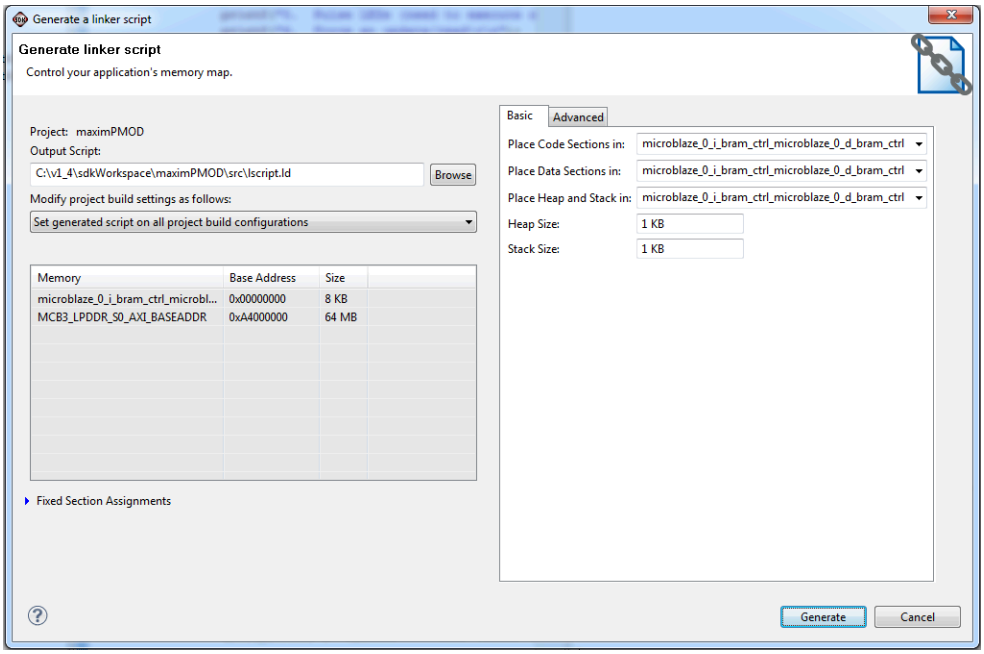
Ahora ya es momento de probar el programa de Temperatura, para ello, colocar el Pmod MAX31723 en el conector J5 y hacer coincidir los pines 1 del conector y del sensor.

En el menú principal, y desde Terminal, escribir "B". Con eso entraremos en el submenú dedicado al Pmod de Temperatura. Probar todas las funciones que tiene. Para variar la temperatura y ver como cambia el valor, colocar el dedo sobre la resistencia R5 (mirar el datasheet del Sensor y entender porqué).

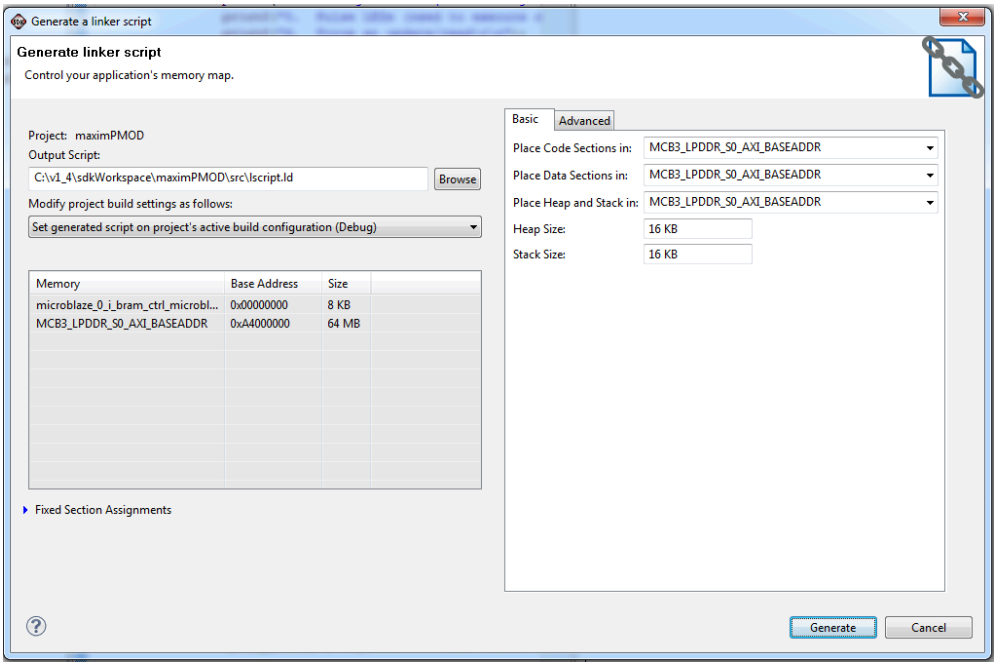
```
MAX31723 Digital Thermometer:

Last temp reading = 26.7 deg C
Low Alarm Setpoint = 0.0 deg C
High Alarm Setpoint = 0.0 deg C
-----------------------------------------------------

1.  Retrieve ( 1)    temp reading
2.  Retrieve (20)    temp reading(s)
3.  Retrieve temp indefinitely
4.  Directly enter low alarm setpoint
5.  Directly enter high alarm setpoint
6.  Toggle display degC / degF

9.  Return to main menu
```

Uso del Linker Script

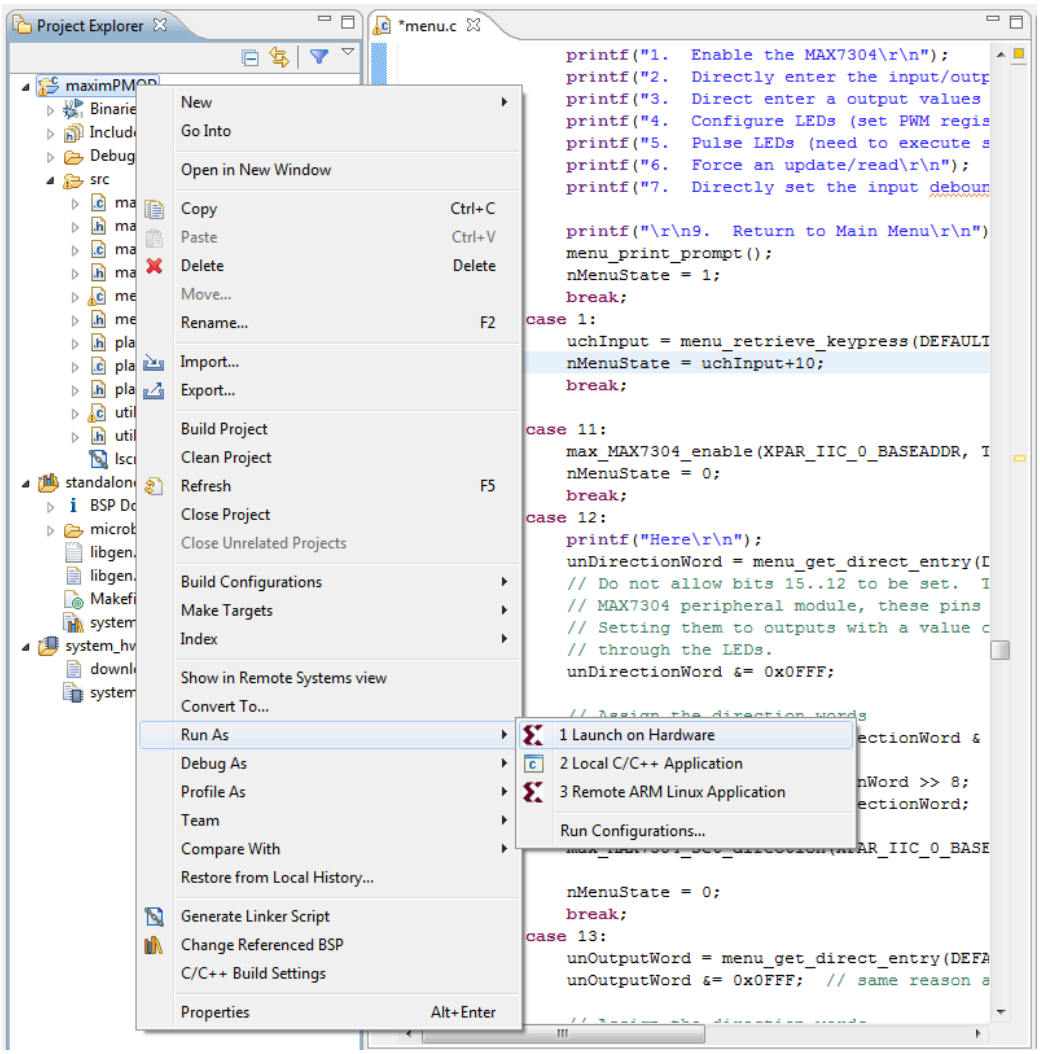
Ya hemos visto como se descarga de modo manual la aplicación software en la DDR, estos pasos los podemos hacer automáticamente usando el Linker Script. Por defecto, si nos ponemos en la carpeta maximPMOD y con el botón derecho accedemos al Linker Script tendremos lo siguiente.



En este menú, podemos ajustar las posiciones de la memoria y la generación de una secuencia de comandos del linker que nos permita lanzar el software desde la DDR. Para nuestro caso, esto se debe establecer en MCB3_LPDDR_S0_AXI_BASEADDR. Definiendo que el modo sea en Debug y que el tamaño de la sección heap/stack sea de 16KB.



Una vez generado el Linker Script, ya no son necesarios todos los pasos realizados desde la Consola del XMD. Ahora, teniendo el hardware programado en la FPGA, podemos ir a maximPMOD -> Run As -> Launch on Hardware.

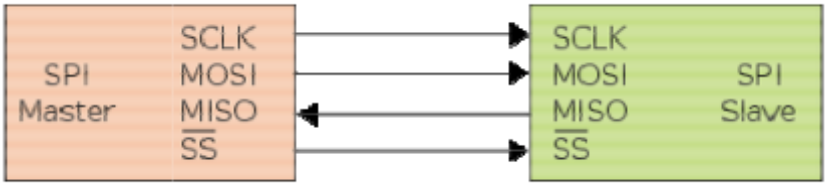


Con lo cual, si miramos en la XMD Console, veremos como se ejecutan los mismo pasos que hemos hecho anteriormente para lanzar el software en la DDR pero de forma automática gracias al Linker Script que hemos modificado.

Protocolo SPI y Funciones de Maxim

El protocolo SPI con el cual estamos trabajando, tiene su nombre de Serial Peripheral Interface bus. Se trata de un nombre estándar proporcionado por Motorola para este bus serie síncrono capaz de operar en modo full-dúplex. Es un bus de cuatro líneas en el que los dispositivos esclavos se acceden mediante líneas de selección (Chip Select).

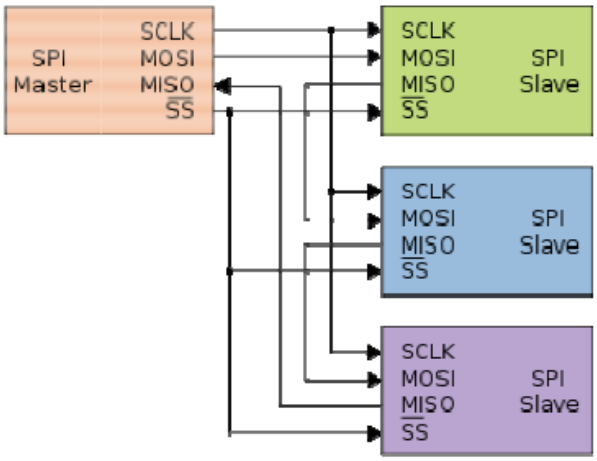
El bus SPI es un estándar de facto, debido a que no existe un estándar formal. Así, diversos tamaños de palabra para la comunicación, protocolos o configuraciones son permitidas dependiendo del dispositivo en uso.



En general, la configuración del bus SPI es la mostrada en la figura. Tiene cuatro señales: SCLK (Serial Clock), MOSI (Master Output, Slave Input), MISO (Master Input, Slave Output) y SS (Slave Select; activa por bajo).

En este caso no existen frecuencias estándar de trabajo, aunque, en general, permite mayores frecuencias que el bus I2C debido a, entre otras cosas, la mayor cantidad de líneas en el bus (pues se separan las lecturas y escrituras en líneas diferentes). Este bus también permite configuraciones multi-máster.

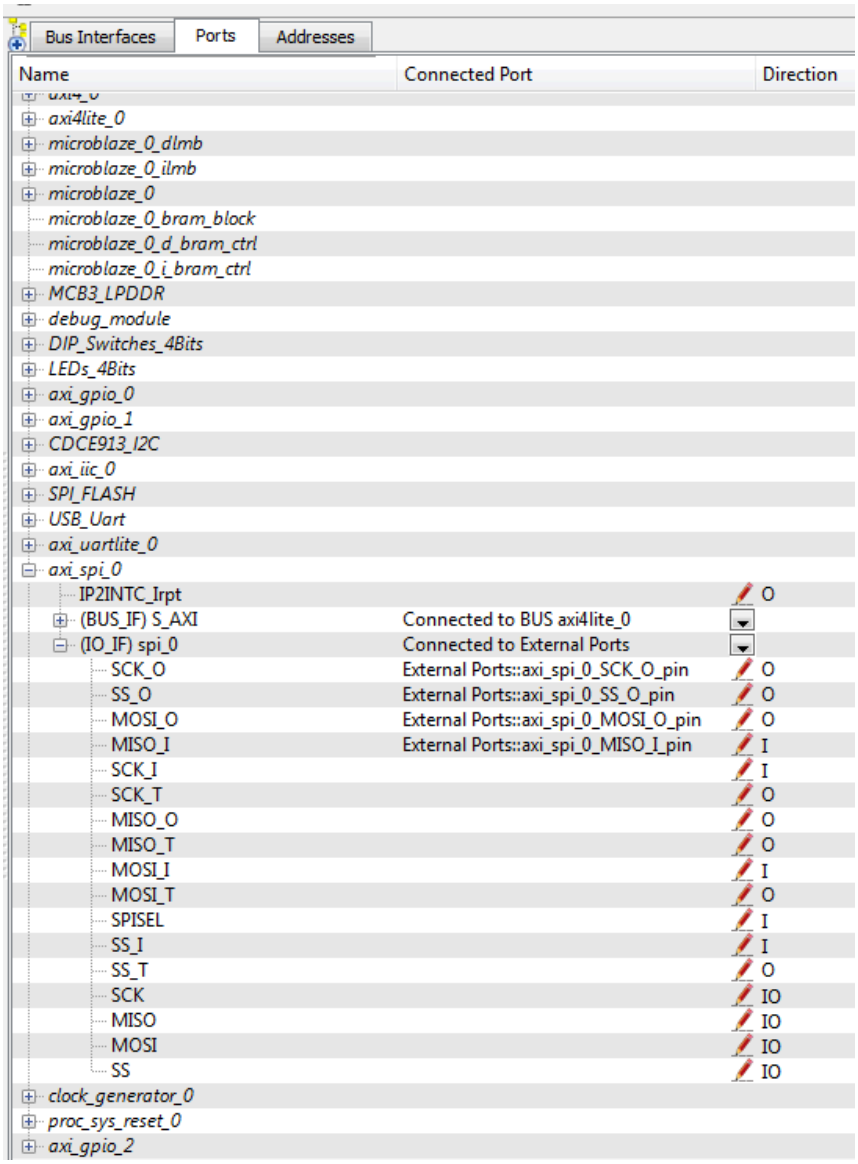
En principio, en la variante básica de este bus, existe una línea SS para cada dispositivo conectado al bus. No obstante, en una configuración con muchos dispositivos esclavos, se puede dar que el número de líneas aumente enormemente, perdiendo una de las mejores características de este tipo de buses.



Para solucionar este problema, este bus también permite una configuración "Daisy Chain", como la mostrada en la figura anterior. En ella, cada uno de los esclavos se conecta con el siguiente con las líneas MOSI y MISO, permitiendo seguir usando una sola línea SS (Slave Select).

Hoy en día, este tipo de bus se usa en una gran variedad de aplicaciones como sensores, dispositivos de control, cámaras, telecomunicaciones, etc. Por ejemplo, el conector JTAG, cuyo uso está extendido para la configuración de las FPGAs, es, esencialmente, una aplicación de un bus de tres líneas de tipo SPI.

Podéis abrir el Proyecto en EDK, carpeta system, y comprobar como se han definido las líneas de SPI comentadas anteriormente. Tendréis que actualizar el proyecto puesto que el de Maxim está realizado para la versión 13.4 de ISE.



Llegados a este punto, se os propone como Ejercicio a realizar, modificar el código ofrecido por Maxim para todos sus Pmods y adecuarlo sólo para el Pmod de Temperatura.

Para entender las funciones que os da Maxim, debéis hacer uso de varios datasheets (MAX31722xMAX31723.pdf, MAX31723PMB1.pdf y axi_spi_ds742.pdf)

Cuando entréis en las funciones, observar en el código los valores de los registros y entender porqué se usan los mismos.

Debéis comprender como se realiza la comunicación observando los Datagramas del módulo de Maxim y distinguiendo correctamente la parte que corresponde al periférico SPI de MicroBlaze y la parte que corresponde al módulo de Maxim.