MÁSTER OFICIAL EN INGENIERÍA ELECTRÓNICA





Curso 2013-2014 José Torres Raimundo García Julio Martos Jesús Soret Adrián Suárez Pedro A. Martínez Abraham Menéndez

VNIVERSITAT (Descola Tècnica Superior d'Enginyeria



Tema 4.- Microprocesador MicroBlaze de Xilinx

Máster Oficial en Ingeniería Electrónica Curso 2013-14



VNIVERSITAT (Description of the second desc



- Introducción a Microblaze.
- □ Buses e interfaces.
- Periféricos de usuario



- MicroBlaze es un bloque microprocesador "software" diseñado para su implementación en FPGAs de Xilinx.
- Presenta un juego de instrucciones reducido (RISC)
- □ Acceso a instrucciones y datos de forma independiente (Harvard)
 - Es habitual que estén en la misma memoria física para facilitar la depuración software.
- La última versión de MicroBlaze que incorpora EDK 10.1 es la 7. Actualmente estamos en la versión 8.4





MicroBlaze frente a otros Microprocesadores Embebidos





□ Arquitectura de MicroBlaze





□ Arquitectura de MicroBlaze



22 January 1073





Arquitectura de MicroBlaze





Características fijas de MicroBlaze

- Presenta un núcleo de 32 bits.
- Buses de direcciones y datos de 32 bits.
- Instrucciones de 3 operandos y 2 modos de direccionamiento (directo e inmediato).
- □ 32 registros de propósito general (R0-R31)
 - □ Se ponen a cero cuando se configura la FPGA.
 - No cambian con las señales de Reset.
- Pipeline de 5 etapas.
 - □ La 1ª instrucción tarda en ejecutarse 5 ciclos de reloj para evitar errores de sincronización, las demás instrucciones se ejecutan en un ciclo de reloj.
- □ Formato Big-Endian (b0...b31) (MSB...LSB)



Características fijas de MicroBlaze

- Unidad aritmético-lógica (ALU)
 - Multiplicación y división entera (hardware)
 - Barrel shifter (desplazamiento bits con sólo una instrucción, hardware)
- Unidad de punto flotante (FPU)
 - Estándar IEEE 754
 - □ Suma, resta, multiplicación, división y comparación.
- Interfaz de depuración hardware (MDM)
- □ Excepciones hardware.
- □ Interfaces FSL.
- Caché de instrucciones y datos.



Características generales de MicroBlaze

Processor					
Processor (CPU) Type	32-bit				
Processor Architecture	MicroBlaze - RISC processor				
MMU	Memory Management Unit				
Processor Frequency	25 Mhz to 200 Mhz				

Peripherals (IP cores)						
Memory	On-Chip RAM, Flash, SRAM, SDRAM, DDR					
Basic IP Peripherals	Timer, Interrupt Controller, UART, GPIO					
Advanced IP Peripherals	TEMAC, CAN, MOST, USB, PCIe, MultiPort Memory Controller, <i>Custom</i>					

Software					
RTOS	Nucleus, ThreadX, uClinux, uC/OS-II, uITRON				
Languages	Assembly, C/C++				
Tools	EDK & Platform Studio, GNU Tools, Eclipse IDE, Nucleus EDGE, Lauterbach T32, Computex F-Sight, LynuxWorks Luminosity, Agilent FPGA DynamicProbe				



Virtex or Spartan FPGA

Microprocesador MicroBlaze de Xilinx

Introducción MicroBlaze

Evolución de MicroBlaze

Fastures	MicroBlaze versions									
reatures	v1 v2 v3		v4	v4 v5		v7				
Pipeline Depth	3	3	3	3	5	3&5	3&5			
Max Integer Perf.	82 DMIPS	125 DMIPS	125 DMIPS	166 DMIPS	240 DMIPS	240 DMIPS	240 DMIPS			
Local Memory	0 or 8 - 64 KB	0 or 8 - 64 KB	0 or 8 - 64 KB	0 or 2 - 128 KB	0 or 2 - 256 KB	0 or 2 - 256 KB	0 or 2 - 256 KB			
Multiplier, Barrel Shifter	option	option	option	option	option	option	option			
Divider	-	option	option	option	option	option	option			
Coprocessor Interface	-	FSL	FSL	FSL	FSL	FSL	FSL			
Instr. & Data Cache	-	0 or 8 - 64 KB	0 or 8 – 64 KB	0 or 2 – 64 KB	0 or 2- 64 KB	0 or 2- 64 KB 648 - 10248 uCache	0 or 2- 64 KB 64B – 1024B uCache			
Cache Interface	-	-	Cache Link	Cache Link	Cache Link	Cache Link	Cache Link			
Floating Point Unit	-	-		single precision 33 MFLOPS	single precision 50 MFLOPS	single precision 50 MFLOPS	single precision 50 MFLOPS			
MMU	-	-			-		Option (MPU or MMU) Full Linux Support			
Debug Interface	ROM monitor	JTAG HW Debug	JTAG HW Debug	Debug + Trace	Debug + Trace	Debug + Trace	Debug + Trace			
Primary FPGA Targets	Virtex II Spartan 2	Virtex II Pro Spartan 2E	Virtex II Pro Spartan 3	Virtex 4 Spartan 3E	Virtex 5	Virtex 5 family Spartan 3 family	Virtex 5 family Spartan 3 family			



Esquema de buses en MicroBlaze





Esquema de buses en MicroBlaze





Processor Local Bus (PLB)





- Processor Local Bus (PLB)
 - □ Estándar de arquitectura de bus CoreConnect de IBM (instrucciones y datos)
 - Está formado por:
 - Maestros (16 máximo)
 - Esclavos (sin límite aunque Xilinx recomienda 16)
 - □ Árbitro de bus.
 - □ Interconexiones de bus.
 - El árbitro de bus recibe las peticiones de bus de los maestros y cede el bus a uno de ellos.
 - Prioridades fija y dinámica.
 - □ Las interconexiones de bus se realizan mediante bridges.



□ Local Memory Bus (LMB)





- □ Local Memory Bus (LMB)
 - Acceso en un solo ciclo de reloj a memoria dedicada (BRAM) dentro de la FPGA
 - □ ILMB: Interfaz de instrucciones.
 - DLMB: Interfaz de datos.
 - Mínimo número de señales de control y protocolo síncrono simple para transferencias de información eficientes.
 - □ Más rápido que el Bus PLB (125 MHz) en todos los dispositivos.



□ Fast Simplex Links (FSL)





- □ Fast Simplex Links (FSL)
 - Interfaces unidireccionales punto a punto dedicados y sin arbitraje basados en memoria FIFO.
 - □ Acceso rápido, 2 ciclos de reloj en Microblaze.
 - Permiten acceso directo a los registros de propósito general desde coprocesadores hardware utilizando instrucciones en C y ensamblador.
 - Un máximo de 8 entradas y 8 salidas.
 - □ Tamaño de FIFO configurable.
 - Reloj de la FIFO síncrono o asíncrono.



Xilinx Cache Link (XCL)





- □ Xilinx Cache Link (XCL)
 - Diseñado para conectar directamente memorias externas a MicroBlaze mediante buffers tipo FSL.
 - Necesario habilitar el uso de caché en MicroBlaze para su uso.
 - Control de hasta 8 memorias externas tipo DDR.
 - Usa un controlador de memorias diseñado por Xilinx.



















- Se generan varios archivos al crear la estructura del periférico:
 - Microprocessor Peripheral Definition (MPD)
 - Define la interfaz del periférico. Tiene la lista de puertos, la lista de conexiones con el bus y los parámetros básicos.
 - Peripheral Analysis Order (PAO)
 - Contiene la lista de archivos HDL que son necesarios para sintetizar el periférico creado, también define el orden en que son compilados y simulados.
 - User_Logic (.vhd)
 - Plantilla en VHDL para que el usuario pueda editar y añadir código. Se suele usar para implementar componentes de usuario que completan el diseño del periférico.
 - □ Core_Usuario (.vhd, .c, .h)
 - □ Son los archivos principales del periférico creado.
 - □ En el .vhd se pueden definir los puertos externos del mismo.
 - □ El .c tiene una estructura para introducir código y el .h las funciones a usar.

Ejercicio

Temporizador e Interrupciones

En este sexto diseño con MicroBlaze vamos a aprender a utilizar un timer para implementar un contador y, adicionalmente, a aprender como funciona el sistema de interrupciones en MicroBlaze.

I desarrollo de este ejercicio nos llevará a crear un diseño hardware con Microblaze utilizando de nuevo BSB con los periféricos habitulaes pero añadiendo también un periférico de tipo XPS Timer que nos establecerá una base de tiempos.

Así mismo, crearemos una aplicación software para gestionar este timer, lo que nos permitirá aprender como funciona el sistema de interrupciones en MicroBlaze. El diagrama de bloques que tendrá esta aplicación es el siguiente.



Creación sistema empotrado - XPS

Siguiendo los pasos del Ejercicio_1, creamos un nuevo proyecto con el nombre Ejercicio_6. 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, usaremos el puerto USB_UART para poder visualizar el flujo de comunicación. También emplearemos los LEDs y los switches, y por último añadiremos un periférico XPS Timer, y lo configuraremos de acuerdo a la siguiente figura:



En la System_Assembly_View podemos observar como, aparte de nuestro periférico Timer, el asistente ha creado otro periférico llamado xps_intc, que es el que va a recoger todas las solicitudes de interrupción generadas por el sistema y gestionarlas de acuerdo a una lista de prioridades para que MicroBlaze las procese de manera adecuada.

Cambiamos el nombre del componente xps_timer_0 por delay. Además, conectamos las siguientes señales del temporizador delay en la pestaña ports:

• CaptureTrig0 a net_gnd (basta seleccionarlo, indica que no tendremos arranque de timer por señal

externa)

• Interrupt a xps_intc_0::Intr (ya debería estar seleccionado. Indica que la interrupción de este periférico debe dirigirse al controlador de interrupciones)

En el periférico xps_intc_0 vemos que la señal de interrupciones consiste en un desplegable en el que nos deja ordenar por prioridades todas las señales de interrupción que recibe este controlador. En este momento conectamos la única que tenemos, con lo que nos quedaría algo así:

8	Interrupt Conne	ction Dialog						×
In Un	errupt Controller	xps_intc_0 💌 pt(s)		⊂o	Show Net Name	s)		Priority
	Instance Name	Port Name		Γ	Instance Name	Port Name	Net Name	Low
1	mb_plb	Bus_Error_Det		0	delay	Interrupt	xps_timer_0_Interrupt	
2	USB_UART	Interrupt						否
3	mdm_0	Interrupt	⇒					~
			\$					₽
								High
							OK Cancel Help	

La salida de este periférico ya debe estar conectada al puerto de interrupciones de MicroBlaze, por lo que finalmente el sistema nos quedará tal y como se muestra en la siguiente figura:

(ev Project Hyrolware (for Protection (for Protection	Ite Device Configuration Debug Sym Configuration Device State Configuration Device State Configuration High-Speed	wlation Window Help ↔⊡&x n PType	Bus Interfaces Parts Addresses Orb donb @ donb	Connected Part	Direction								(E
Cock, Rese Communic Cock, Rese		++⊡ & x n IP Type	Bus Interfaces Ponts Addresses Name	Connected Port	Direction								(C
X P Catalog Catalog Catalog Catalog Description S EDK Install Analog Analog Analog Analog Analog Communic Communic Communic DMA and Defan	IP Versio II ridge set and Interrupt ications High-Speed	⇔⊡ & ×	Bus Interfaces Ports Addresses Name dimb imb mb,pib minupib	Connected Port	Direction								Į.
Communic Communic Communic Communic Communic Communic Communic DMA and T	P Versio P Versio P Versio P Versio P Versio	п № Туре	Name dimb di	Connected Port	Direction								
Description E EDK Install Analog Arbitrotic Bus and Brit Clock, Rese Communic Communic DMA and Defen	IP Versio Al Indge set and Interrupt Lication High-Speed	n 🏼 P Type	⊕ dimb ⊕ imb ⊕ mb_pib ⊕ mippib			Range	Class	Frequency(Hz) Reset Polar	ty Sensitivity		P Tune	Differential Polarity	
EDK Install EXC Install Analog Analog Arithmetic Bus and Bri Clock, Rese Communic DMA and T DMA and T	II vesio ridge set and Interrupt ication High-Speed	n propre	imb mb_pib dentification						.,		ste lesh vill		
EDK Install Analog Analog Arithmetic Bus and Bri Clock, Rese Communic DMA and T DAhun	r c Iridge set and Interrupt ication High-Speed		⊕ and ⊕ da_da ⊕ eximpleme 0								the lesh still		
Analog Arithmetic Bus and Bri Clock, Rese Communic DMA and T DAtum	c Iridge set and Interrupt ication High-Speed		C microbleme ()								and all off		
Arithmetic Bus and Bri Clock, Rese Communic DMA and T DMA and T	c Iridge set and Interrupt ication High-Speed										M bio_veo		
Bus and Bri Clock, Rese Communic Communic DMA and T DMA and T	Iridge set and Interrupt ication High-Speed						0/7				M unicioniaze		
Clock, Rese Communic Communic DMA and T DMA and T	set and Interrupt ication High-Speed		MB_RESET	proc_sys_reset_0::M8_Reset	<u></u> 1		RST						
Communic Communic DMA and T	ication High-Speed		- DBG_STOP		1								
Communic DMA and T Debug			- MB_Halted		10								
DMA and T Dehug	ination Low-Speed		- MB_Error		/ 0								
Contra and a	Timer		WAKEUP		/1	[0:1]							
	1110		- SLEEP		/0								
o rock p			- DBG WAKEUP		10								
t PPGA Kecol	orngutation		LOCKSTEP MASTER OUT		70	10-40951							
General Pur	urpose IU		LOCKSTER SLAVE IN		1	10-40951							
IO Modules	e		LOCKETER OUT		10	0.40051							
Interprocess	essor Communication		- LOCKSTEP_COT	6	20	[004030]							
Memory an	and Memory Controller		(BO2'IE) DOWR	Connected to BUS dimp									
an 🕒 PCI			I (BUS_IF) ILMB	Connected to BUS limb									
Peripheral C	Controller		⊕ (BUS_IF) DPLB	Connected to BUS mb_plb	-								
Processor			⊕ (BUS_IF) IPL8	Connected to BUS mb_plb									
IN LISER			(BUS_IF) INTERRUPT	Not connected to BUS or External Ports									
0.1926			- INTERRUPT						LEVEL HIGH				
C Mail ratio	-		- Ind bram						-		tr bram block		
- Vernication	in in in		dissh catir								w lesh bram i		
E Noto and a	a image Processing		ilesh catle								and lesh bran i		
Project Local Pi	PCores		D and a								H mouth		
			a wento								N INSU		
			E xbs_anc_o						and provide		M abs utc		
			- Intr	L to Ht sps_timer_0_interrupt		[1:0]	INTERMUPT		EDGE_RISING				
			- Irq	microblaze_0::[INTERRUPT]::INTERRUPT	10		INTERRUPT		EDGE_RISING				
			D/P_Switch_4Bits								🗙 xps_gpio		
			⊕ LEDs_48its								📩 xps_qpio		
			😑 delay								ir aps timer		
			- Capture Trip0		/1								
x			CartureTripl		71								
			Generate()ut)		10								
			Countrol		10								
			- GenerateOutL		10								
			PWMD		10								
			- Interrupt	xps_intc_0::Intr	20		INTERRUPT		EDGE_RISING				
			Freeze		<u>/</u> 1								
			⊕ USB_UART								📩 xps_uartite		
			⊕ clock_generator_0								📩 clock,gene.		
			+ proc sys reset 0								TO DIDC SIS ID.		
			1										
<	*	•	Legend										
			Master Slave Master/Slave Target (In	tiator Connected OUnconnected M Monitor									
Search IP Catalog:	s	Clear	WProduction SUcense (paid) SUcense (er	Cocal Pre Production P38eta PDevelo	opment								
			Superseded ODiscontinued										
🍪 Project 🛞	IP Catalog		🗵 Design Summ	ary 🔝 😸		Graphical Design Vie	N	🖸 🚷	System Assembly View	×			
Console													
Copied C:/)	/Xilinx/14.4/ISE DS/EDK/data	/xflow/bitgen sparts	m6.ut to etc directory										

Una vez realizados todos estos cambios, procedemos a generar el bitstream, a exportarlo a SDK, generamos el correspondiente BSP y procedemos a implementar el test de periféricos para comprobar que todo funciona adecuadamente.

Antes de implementar nuestro código, debemos de entender el funcionamiento del periférico XPS_Timer. Para ello, busca la documentación proporcionada por Xilinx y localiza en ella la descripción de los registros, prestando especial atención al registro de control/status (TCSR0). Compara los parámetros que aparecen en ese registro con lo que ya conoces sobre el funcionamiento de un Timer.

http://www.xilinx.com/support/documentation/ip_documentation/xps_timer.pdf

Estudia en el system.mss del SDK los drivers creados por Xilinx para la utilización del timer. Estudia con atención las funciones, los parámetros, y los ficheros de ejemplo.



Ahora vamos a crear una nueva aplicación software, cuyo código nos viene dado en el fichero temporizador.c

```
#include "xparameters.h"
#include "xbasic_types.h"
#include "xgpio.h"
#include "xtmrctr.h"
#include "xintc.h"
#include "stdio.h"
XTmrCtr MyTimer, *MyTimerPtr;
XGpio MyLeds, *MyLedsPtr;
int LedBit;
int count = 0;
int flag 5 sec = 0;
void timer int handler(void *TimerPtr)
{
    XGpio DiscreteWrite (MyLedsPtr, 1, 1 << LedBit);
    LedBit++; if (LedBit == 4) LedBit = 0;
    count++;
    if (count\$5 == 0) flag 5 sec = 1;
}
int main (void)
{
    XIntc MyIntc, *MyIntcPtr;
    // Configure interrupt controller
   MyIntcPtr = &MyIntc;
    if (XIntc Initialize(MyIntcPtr, XPAR INTC 0 DEVICE ID) !=
XST SUCCESS)
    return XST FAILURE;
    XIntc Connect(MyIntcPtr, 0, timer int handler, MyTimerPtr);
    XIntc Start(MyIntcPtr, XIN REAL MODE);
    XIntc Enable(MyIntcPtr, 0);
    // After configuring the controller, interrupts can be enables
    microblaze enable interrupts();
    // Initialize timer
    MyTimerPtr = &MyTimer;
    if (XTmrCtr_Initialize(MyTimerPtr, XPAR_DELAY_DEVICE_ID) !=
XST SUCCESS)
    return XST FAILURE;
    XTmrCtr SetResetValue(MyTimerPtr, 0, 5000000);
    XTmrCtr SetOptions (MyTimerPtr, 0,
        XTC DOWN COUNT OPTION + XTC AUTO RELOAD OPTION +
XTC INT MODE OPTION);
```

```
// Start timer
XTmrCtr_Start(MyTimerPtr, 0);
// Initialize GPIO
MyLedsPtr = &MyLeds;
if (XGpio_Initialize(MyLedsPtr, XPAR_LEDS_4BITS_DEVICE_ID) !=
XST_SUCCESS)
return XST_FAILURE;
XGpio_SetDataDirection(MyLedsPtr, 1, 0x0);
XGpio_DiscreteWrite(MyLedsPtr, 1, 0x0);
while(1) {
if (flag_5_sec==1)
flag_5_sec = 0;
xil_printf("Han pasado 5 segundos\n\r");
}
```

Este código crea una base de tiempos de 1 segundo, y va encendiendo los leds de manera secuencial cada segundo. Además, cada 5 segundos saca un mensaje por hyperterminal.

El código compila sin errores, pero su funcionamiento no es correcto dado que tiene dos errores, uno respecto al propio lenguaje C, y otro respecto a la forma de gestionar la interrupción. Además, la base de tiempos no es exactamente de 1 segundo. Localiza estos errores y corrígelos.

Como ejercicio adicional, se propone realizar un proyecto en el que el controlador de interrupciones controle tanto las interrupciones de un timer como de dos switches independientes. La cuenta en segundos aparecerá en el hyperterminal, y uno de los switches controlará el sentido de la cuenta mientras que el otro actuará como start/stop.

Ejercicio

Integración de un Core como periférico

Si tenemos un Core diseñado previamente en ISE o con Simulink, podemos añadirlo a EDK según se va a explicar en este ejercicio.

menudo, creamos Cores propios que hemos usado en diferentes proyectos y que podemos usar en cualquier nuevo desarrollo. El objetivo de este ejercicio es conocer como se puede integrar dicho. Core dentro de un proyecto diseñado con MicroBlaze. Para dicha integración, usaremos el asistente para periféricos que incluye EDK.

El primer paso será crear un Core mediante la aplicación Xilinx Core Generator que incluye ISE y que nos genera un archivo .ngc

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



Creación de un Core

Como hemos comentado antes, vamos a usar la aplicación Xilinx Core Generator para crear un periférico que nos permita realizar la multiplicación de dos entradas de 16 bits y nos genere un resultado de 32 bits.

Desde el Menú de Xilinx, buscamos la aplicación comentada anteriormente. Se encuentra dentro de la carpeta de 32 o 64 bits (según corresponda) en ISE Design Tools.



Seleccionando File->New Project escogemos donde lo queremos guardar. Lo habitual es crear una carpeta denominada Cores y almacenarlos todos ahí. Darle, por ejemplo, el nombre de mult.

Ahora debemos especificar con que FPGA vamos a utiliza el Core.

Y Project Options			? X
Part .	Part		
Generation	Select the part for	your project:	
- Advanced	Fa <u>m</u> ily	Spartan6	-
	De <u>v</u> ice	xc6slx9	-
	P <u>a</u> ckage	csg324	•
	Speed Grade	-2	•
	_		
	<u><u> </u></u>	K <u>C</u> ancel Appl	y <u>H</u> elp

El resto de opciones las dejamos por defecto.

Una vez creado el Proyecto, debemos escoger que función es la que va a realizar. Antes de esto, explorar las diversas opciones de periféricos que nos presenta el CoreGenerator. Para implementar un core con la funcionalidad de multiplicador, entramos en Math Functions, luego en Multipliers y seleccionamos el core Multiplier.



Es interesante leerse y comprender la función que realiza y como la realiza. Podéis abrir los diferentes documentos y conocer en profundidad este Core. Haciendo doble clic, abriremos el asistente del Core seleccionado.

🍕 Multiplier		Contract Contract Res and some Contract Contract	
Documents <u>V</u> iew			
IP Symbol 🗗 🛪	Logi CKRE	Multiplier	xilinx.com:ip:mult_gen:11.2
	Component Name : Multiplier Type Parallel Multipl Constant-Coeff	multiplier lier ficient Multiplier	
A[15:0] → → P[31:0] B[15:0] →	Input Options Port A Data Type : Width :	Signed • 16 Range: 264	
CE → SCLR →	Port B Data Type : Width :	Signed The Range: 264	
	(Delation)		
🌂 IP Symbol 🌂 Resource Estimates	Datasheet	< <u>B</u> ack Page 1 of 3 Nex	t > <u>G</u> enerate <u>C</u> ancel <u>H</u> elp

Mantenemos el nombre del core. Definimos la anchura de las señales A y B a 16 bits, y comprobamos en la pestaña IP symbol las entradas y salidas que tendrá nuestro core. Al decirle que utilize un multiplicador dedicado en lugar de uno basado en LUTs, en interesante observar como se utilizan exclusivamente XtremeDSP slices y ninguna LUT.

🌾 Multiplier			
Documents <u>V</u> iew			
Resource Estimates P × Resource Estimates LUT6s 0 XtremeDSP slices 1 BRAMS 0 Additional Information Please note that the LUT resource estimate does not include SRLs. Resource counts may not reflect true post-map resource usage when a custom output width is used and the output product MSB is less than full-precision MSB	Parallel Multiplier Options Multiplier Construction Multiplier Construction : Use Optimization Options Speed Optimized Optimizes the multiplier Area Optimized Optimizes the multiplier embedded multipliers a	Multiplier e Mults r for performance using as many embedded r for embedded multiplier resources by sp and slice logic	xilinx.com:ip:mult_gen:11.2
🍕 IP Symbol 🍕 Resource Estimates	Datasheet	< Back Page 2 of 3 Next 2	> <u>G</u> enerate <u>C</u> ancel <u>H</u> elp

Dejando el resto de opciones por defecto, finalizamos el asistente y cerramos Core Generator con nuestro Core terminado.

Para añadir dicho Core a MicroBlaze, seguiremos los siguientes pasos:

Lo primero será crear el hardware de MicroBlaze usando el puerto serie habitual, y sin necesidad de usar ningún periférico más. Creamos las aplicaciones de test y generamos nuestro sistema empotrado.

Una vez generado el bitstream y comprobado que no tenemos errores, en el menú Hardware escogemos Create or Import Peripheral.

Seleccionamos la creación de plantillas para un nuevo periférico y lo almacenamos dentro del directorio de nuestro Ejercicio_7. Le damos el nombre de mi_multiplicador y escogemos el bus PLB para su conexión con MicroBlaze.

Para el protocolo IPIF, seleccionamos la posibilidad de usar una FIFO para la lectura y escritura de los datos del Core del multiplicador. También seleccionamos la inclusión de un timer para su control (Include data phase timer) y deseleccionamos el uso del registro.

Create Peripheral IPIF (IP Interface) Services Indicate the IPIF services required by your peripheral. Your peripheral will be connected to the PLB (v4.6) interconnect through co way to implement the interface between the PLB interconnect and the user	rresponding PLB IP Interface (IPIF) modules, which provide you with a quick logic. Besides the standard functions like address decoding provided by the
seve intermedule, the withd tool also others other commonly used services	Sara configurations to simplify the implementation of the design. Slave service and configuration Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (PLB slave interface will always be included). Software reset User logic software register Read/Write EIFO User logic memory gpace Interrupt control Master service and configuration Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected). User logic master
More Info	< Back Next > Cancel

Dejamos el resto de opciones por defecto hasta la ventana de Peripheral Implementation Support, donde activamos la pestaña de Generate Template Drivers.



Ya tenemos creada la base del periférico, como queremos que dicho periférico funcione con los bloques FIFO creados, debemos modificar ciertas líneas del archivo user_logic.vhd. Abrimos dicho archivo y añadimos lo siguiente:

```
169
170
       --USER signal declarations added here, as needed for user logic
171
172
     component multiplier
173
       port (
174
         clk: IN std logic;
        a: IN std_logic_VECTOR(15 downto 0);
175
         b: IN std logic VECTOR(15 downto 0);
176
177
         p: OUT std logic VECTOR(31 downto 0));
178
     end component;
179
180
191
       --USER logic implementation added here
192
193
194
     multiplier O : multiplier
```

```
194 multiplier_0 : multiplier
195      port map (
196      clk => Bus2IP_Clk,
197      a => WFIF02IP_Data(16 to 31),
198      b => WFIF02IP_Data(0 to 15),
199      p => IP2RFIF0_Data);
200
```

No debemos cambiar el nombre "multiplier" puesto que ese es el nombre que le da el Core Generator al multiplicador. También debemos comentar la siguiente línea:

272
273 --IP2RFIF0_Data <= WFIF02IP_Data;
274</pre>

Con todo esto hemos conseguido que los valores que salgan de la FIFO pasen por nuestro multiplicador y que el resultado de dicha multiplicación se envíe de nuevo a la FIFO antes de enviarse al bus PLB.

Ahora deberemos incluir nuestro Core para que entienda estas líneas y funcione. Para ello, volvemos a abrir el Asistente para la Creación de Periféricos y seleccionamos la opción Import existing peripheral.

Importamos el periférico creado, con especial atención al número de versión, escogiendo según sea el periférico la opción de HDL o Netlist. Con la primera opción importaríamos periféricos creados en VHDL y con la segunda periféricos creados con Core Generator o con Simulink. En nuestro caso marcamos ambas opciones.

😵 Import Peripheral	? <mark>x</mark>
Source File Types Indicate the types of files that make up your peripheral.	*
Indicate the types of files that make up your peripheral.	
<u> H</u> DL source files (*.vhdl, *.v, *.vh)	
✓ Netlist files (*.edn, *.edf, *.ngc, *.ngo)	
Documentation files (*.pdf, *.doc, *.txt)	
More Info	Cancel

Seleccionamos el .pao de nuestro periférico para que entienda la estructura creada en el mismo.

8 Import Peripheral	? ×
HDL Source Files Indicate how this tool should locate the HDL files that make up your peripheral.	X
Use data (*.mpd) collected during a previous invocation of this tool	Browse
How to locate your HDL source files and dependent library files	
Use an <u>XST</u> project file (*.prj) This tool will input the HDL file-set and the logical libraries they are compiled into from the appropriate lines in the project file.	
	Browse
Use existing Peripheral Analysis Order file (*.pao)	
C: \/\iinx\Ejercicio_7\pcores\mi_multiplicador_v1_00_a\data\mi_multiplicador_v2_1_0.pao	Bro <u>w</u> se
Browse to your HDL source and dependent library files (*.vhd, *.vhd, *.v, *.vh) in next step	
More Info	Cancel

Seleccionamos la opción de conectar el periférico como esclavo al bus PLB.

Bus Interfaces Identify the bus interfaces supported by your periphere	al.	\$
A bus interface is a group of related interface ports distinguing by your peripheral or indicate if there is no applicable bus int	ished by a bus standard (i.e. PLBV46, DCR, or FSL). Select the bus terface.	interface(s) supported
Select bus interface(s)		
AXI bus interface		
AXI4Lite	AXI4	
 Master 	 Master 	
Slave	Slave	
Processor Local Bus (version 4.6) interface	Fast Simplex Link bus interface	
PLBV46 Master (MPLB)	FSL Master (MFSL)	
Generate burst	ESI Slave (SESI.)	
V PLBV46 Slave (SPLB)		
Device Control Register bus interface		
DCR Slave (SDCR)		

Definimos el espacio de registros:

😵 Import Peripheral	? <mark>×</mark>
SPLB : Parameter Define the SPLB bus interface parameter(s) for this peripheral.	\$ \$
The SPLB bus interface is defined by a predefined set of ports and parame automatically done the selections for you. Otherwise check off the values.	ters. If your peripheral follows the standard naming conventions, this tool has
Register Space	
Parameter determine base address:	C_BASEADDR
Parameter determine high address:	C_HIGHADDR
Memory Space	
e Address Parame h Address Parame Cacheable	A <u>d</u> d Remo <u>v</u> e
More Info	<back next=""> Cancel</back>

Cambiamos la familia que viene por defecto a spartan6 y añadimos el .ngc creado con Core Generator que incluye nuestro periférico.

8	Import Peripheral			<u> २</u>					
	Parameter Attributes Identify the parameters that require	specia	al handling.	×;					
	Select the parameter on the left and fill i the system it is instantiated in.	t the parameter on the left and fill in the attribute values to the right. These attributes help the various tools in EDK to integrate this peripheral into ystem it is instantiated in.							
	- List User Parameters only -		ributes:						
	C_INCLUDE_DPHASE_TIMER	Pa	arameter Name	C_FAMILY					
		Di	ata Type	string					
		D	efault Value	spartan6					
			Display <u>a</u> dvanced	d attributes					
	More Info			< <u>B</u> ack <u>N</u> ext > Cancel					

1	Import Peripheral	? ×
	Netlist Files Identify black-box netlists associated with this peripheral.	*
	Use the buttons on the right to locate any netlist files for 'black-box' components instantiated in your peripheral.	
	C:\Xilinx\cores\multiplier.ngc	Select Files
		<u>R</u> emove Files
_		
	More Info	Cancel

Finalizamos y ya tendremos un nuevo periférico creado a partir de un Core de ISE. Los mismos pasos se harían para incluir un periférico creado en VHDL.

Ya podemos añadir el bloque IP creado a nuestro sistema embebido. Conectándolo al bus, y dándole un tamaño de 64K dentro del mapa de memoria de MicroBlaze. Generamos el bitstream y comprobamos que no hay errores.

Sólo nos queda generar una aplicación software en C para comprobar su funcionamiento. Para ello, creamos en SDK una nueva aplicación software y añadimos los archivos mi_multiplicador.c y mi_multiplicador.h que se han creado automáticamente en la carpeta ..\Ejercicio_7\drivers\mi_multiplicador_v1_00_a\src

Reemplazamos el código que contiene por el siguiente:

#include "xparameters.h"
#include "xbasic_types.h"
#include "xstatus.h"
#include "mi_multiplicador.h"

Xuint32 *baseaddr_p = (Xuint32 *)XPAR_MULTIPLICADOR_0_BASEADDR;

int main (void) {

Xuint32 i; Xuint32 temp; Xuint32 baseaddr;

xil_printf("%c[2J",27);

// Comprobamos que el periférico existe

```
XASSERT_NONVOID(baseaddr_p != XNULL);
baseaddr = (Xuint32) baseaddr_p;
xil_printf("Test Multiplicador\n\r");
```

```
// Resetea la lectura y la escritura de las FIFOs
MI_MULTIPLICADOR_mResetWriteFIFO(baseaddr);
MI_MULTIPLICADOR_mResetReadFIFO(baseaddr);
```

```
// Escribe valores en la FIFO
for(i = 1; i <= 4; i++ ) {
    temp = (i << 16) + i;
    xil_printf("Valores: 0x%08x \n\r", temp);
    MI_MULTIPLICADOR_mWriteToFIFO(baseaddr,0, temp);
}</pre>
```

```
// Lectura del valor de la FIFO
for(i = 0; i < 4; i++){
    temp = MI_MULTIPLICADOR_mReadFromFIFO(baseaddr,0);
    xil_printf("Multiplicacion: 0x%08x \n\r", temp);
}</pre>
```

```
// Resetea la lectura y la escritura de las FIFOs
MI_MULTIPLICADOR_mResetWriteFIFO(baseaddr);
MI_MULTIPLICADOR_mResetReadFIFO(baseaddr);
xil_printf("Fin Test\n\n\r");
```

```
// Permanece en un bucle infinito
while(1){
    }
}
```

Examina la salida del hyperterminal al ejecutar este código. ¿Entiendes lo que está pasando? ¿Cómo funciona la FIFO? ¿Por qué la última multiplicación no sale bien?

Se propone la generación de otro core, en este caso un comparador de 16 bits, y su introducción en el proyecto, de modo que compare los dos valores que se van a multiplicar y que indique en cada caso si el primer operando es mayor que el segundo o no.