Diseño de módulos en VHDL que permitan adicionar los periféricos PMB (Peripheral Module Bundle) de Digilent al microprocesador embebido MicroBlaze, usando el sistema de desarrollo Spartan 3A DSP de Xilinx

William Méndez Ortiz
Carlos Augusto Nieto Cubides

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
Bucaramanga, 2009

Diseño de módulos en VHDL que permitan adicionar los periféricos PMB (Peripheral Module Bundle) de Digilent al microprocesador embebido MicroBlaze, usando el sistema de desarrollo Spartan 3A DSP de Xilinx

William Méndez Ortiz
Carlos Augusto Nieto Cubides

Trabajo de grado para optar el título de Ingeniero Electrónico

Director:
Mie (c) Carlos Augusto Fajardo Ariza

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
Bucaramanga, 2009

Este proyecto está dedicado a nuestras familias quienes aportaron su esfuerzo y apoyo para la contribución de nosotros como ingenieros y a nuestros amigos por su acompañamiento durante nuestra carrera.

William y Carlos

AGRADECIMIENTOS

Un profundo y especial agradecimiento a nuestras familias por su comprensión y apoyo durante toda nuestra carrera

Agradecemos a todos aquellos que han contribuido en nuestra formación como ingenieros y como personas; Un agradecimiento sincero a nuestro director Carlos Augusto Fajardo Ariza por su incondicional apoyo, acompañamiento y guía durante la realización de éste proyecto; Al profesor Jorge Hernando Ramón Suárez por su experiencia, continuo apoyo y motivación para llevar a cabo éste proyecto.

RESUMEN

TÍTULO: DISEÑO DE MÓDULOS EN VHDL QUE PERMITAN ADICIONAR LOS PERIFERICOS PMB (PERIPHERAL MODULE BUNDLE) DE DIGILENT AL MICROPROCESADOR EMBEBIDO MICROBLAZE, USANDO EL SISTEMA DE DESARROLLO SPARTAN 3A DSP DE XILINX[1].

AUTORES: Méndez Ortiz William y Nieto Cubides Carlos Augusto[2].

PALABRAS CLAVES: Embedded Systems, MicroBlaze, PMB de Digilent, FPGA, VHDL, Embedded Development Kit.

DESCRIPCIÓN:

A través de este proyecto se generó un aporte en el diseño de los sistemas embebidos, basados en una de las tendencias tecnológicas más claras respecto al hardware como lo son las FPGAs. El enfoque de diseño elegido fue un procesador IP, al cual se le adicionaron módulos descritos en VHDL, todo esto implementado en una FPGA, esto con el fin de aportar en el diseño de sistemas embebidos más flexibles y económicos, permitiendo contribuir en la solución de problemas que requieran aplicaciones embebidas.

Se utilizó una metodología en la que se trabaja con herramientas CAD para el diseño de aplicaciones embebidas. Esta metodología permitió la descripción en VHDL de los módulos que controlan los periféricos PMB (*Peripheral Module Bundle*) de Digilent, la adición de estos al microprocesador IP MicroBlaze configurado a través del software EDK y la validación a través de aplicaciones por software.

Por medio del trabajo realizado en este proyecto se pretende hacer una contribución en la apropiación tecnológica del desarrollo de *Embedded Systems* de bajo costo y gran aplicabilidad, corroborando los beneficios del trabajo con FPGAs y adicionalmente pretende hacer un aporte de material pedagógico que pueda servir como apoyo para procesos de enseñanza y aprendizaje en el diseño de *Embedded Systems*.

[1]Trabajo de Grado

[2]Facultad de Ingenierías Fisicomecánicas. Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones. Director: Mie(c) Carlos Augusto Fajardo Ariza.

ABSTRACT

TITLE: MODULE DESIGN IN VHDL TO ALLOW THE ADDITION OF THE PERIPHERAL PMB (PERIPHERAL MODULE BUNDLE) OF DIGILENT TO MICROBLAZE EMBEDDED MICROPROCESSOR, USING THE DEVELOPMENT SYSTEM SPARTAN 3A DSP DEVELOPMENT SYSTEM OF XILINX[3].

AUTHORS: Méndez Ortiz William y Nieto Cubides Carlos Augusto[4].

KEYWORDS: Embedded Systems, MicroBlaze, PMB of Digilent, FPGA, VHDL, Embedded Development Kit.

DESCRIPTION:

Through this project generated a contribution in the design of embedded systems, based on technology trends clearer with regard to hardware such as FPGAs. The design approach chosen was a processor IP, to which we add modules described in VHDL, all implemented in a FPGA, this in order to contribute to the design of embedded systems more flexible and cost, allowing help in solving problems demanding embedded applications.

We used a methodology that works with CAD tools for designing embedded applications. This methodology allowed the VHDL description of the modules that control the peripherals PMB (Peripheral Module Bundle) from Digilent, adding these to the IP microprocessor MicroBlaze configured through EDK software and validation through software applications.

Through the work realized in this project is pretending make a contribution in the technology appropriation of Embedded System development of low cost and wide applicability, confirming the benefits of working with FPGAs and additionally seeks make a contribution of pedagogical materials that can serve as support for teaching and learning processes in the design of Embedded Systems.

[3]Degree Work

[4]Faculty of Physical-Mechanical. Engineering Electrical, Electronic and Telecommunications School. Director: Mie(c) Carlos Augusto Fajardo Ariza.

Índice de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1

1. Conceptos básicos

1.1 Procesadores IP *softcores*

Un Procesador IP (*Intellectual Property Core*) también conocido como Bloque IP, es un módulo que tiene una funcionalidad y una interfaz definida y puede ser integrado en otros diseños (Reutilizable), básicamente es un diseño especificado descrito en un lenguaje HDL (*Hardware Description Language*). El término IP (*Intellectual Property*) se deriva de la licencia de la patente y los derechos de copia de los códigos fuentes como derechos de propiedad intelectual que existe en el diseño e inicialmente fueron concebidos como método para reutilización de Hardware (Bibliotecas de Bloques en HDL). Los Procesadores IP (*IP Cores*) se clasifican en: *Softcores*, *Hardcores* y *Firmcores* [6]. Los procesadores IP *softcores* son procesadores que pueden ser implementados totalmente usando síntesis lógica, usualmente lenguaje HDL, generalmente son implementados en un dispositivo de lógica programable como un FPGA. Esto hace que los procesadores IP *softcores* a diferencia de los *Hardcores* y *Firmcores* sean flexibles, ya que sus parámetros pueden ser modificados en cualquier momento reprogramando el dispositivo (FPGA[5], CPLD).

[5] Field Programmable Gate Array, es un dispositivo reprogramable que permite implementar diversos circuitos digitales.

1.2 El microprocesador MicroBlaze de Xilinx

El microprocesador MicroBlaze es un procesador IP, *Softcore*, diseñado para trabajar específicamente en *embedded systems*[6], el diagrama de este microprocesador se muestra en la figura 1.

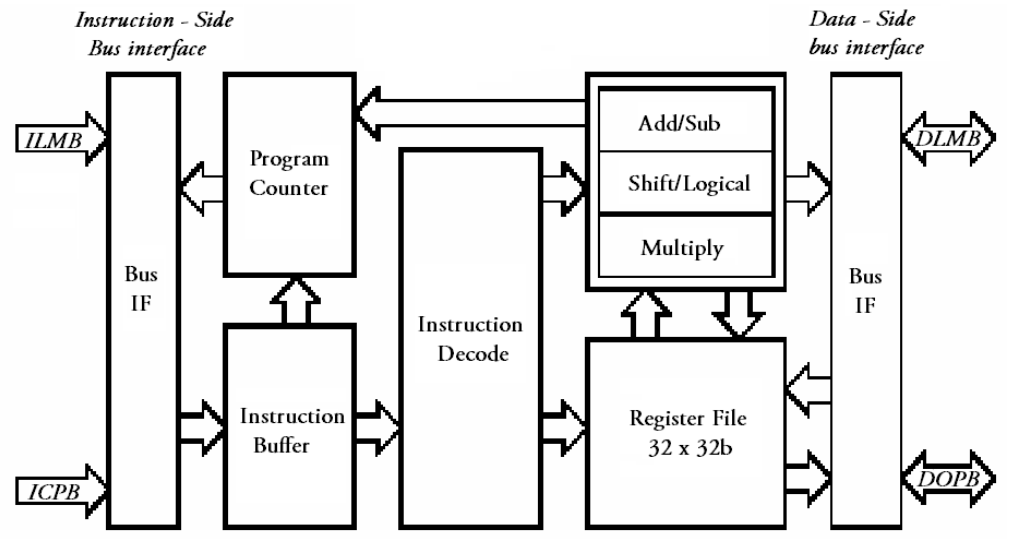

Figura 1: Diagrama de Bloques MicroBlaze [2]

Algunas de las características del MicroBlaze son [1]:

- Arquitectura Harvard, esta arquitectura maneja una memoria de datos y una memoria de instrucciones.
- Procesador RISC[7] (*Reduced Instruction Set Computer*).
- Registros internos de 32 x 32 bits.
- Permite la ejecución de instrucciones de manera segmentada (*Pipelining*).
- 32 registros de propósito general y más de 18 registros de propósito especial.

[6] Combinación de Hardware y software, y periféricos, diseñados para realizar una función específica.
[7] Consta de unas pocas instrucciones muy simples y altamente optimizadas (realizan tareas básicas).

- 87 instrucciones para realizar operaciones lógicas, aritméticas, comparaciones, etc.
- Instrucciones de 32 bits con 3 operandos y 2 modos de direccionamiento.
- Utiliza un bus específico LMB (*Local Memory Bus*) para acceder a las memorias de Bloque disponibles en la FPGA y un bus estándar PLB (*Processor Local Bus*) para conectar memoria externa y periféricos.
- Soporta operaciones con Punto Flotante.
- Flexibilidad.
- Diseñado para trabajo sobre Embedded Systems.

Éste microprocesador permite ser configurado a través del software EDK (*Embedded Development Kit*), permitiendo controlar y direccionar de manera práctica el MicroBlaze. Igualmente, al ser un procesador diseñado para *Embedded Systems*, se integran periféricos a este procesador que permiten aprovechar esta característica y puedan ser útiles en aplicaciones de este tipo (figura 2).

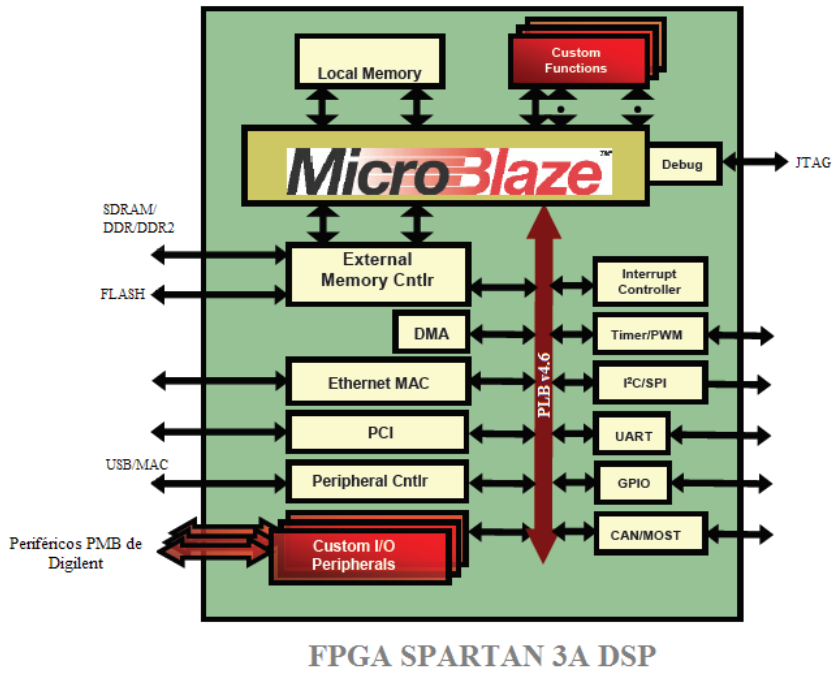

Figura 2: Esquema de conexión del MicroBlaze a través del software EDK.

1.3 La plataforma Spartan 3A DSP de Xilinx

La plataforma *Spartan 3A DSP 1800A* de *Xilinx* (figura 3) es un sistema de desarrollo que permite el diseño de *Embedded Systems* basados en el FPGA *Xilinx Spartan 3A DSP*. El diseño de los *Embedded Systems* se realiza a través de las herramientas CAD[8] de Xilinx, las cuales ofrecen muchas facilidades al momento de implementar un sistema específico.

Figura 3: Spartan 3A DSP 1800A de XILINX

Adicionalmente esta plataforma facilita el diseño e implementación de aplicaciones haciendo uso de algunos periféricos específicos con los que cuenta [3] en la figura 4 se muestra un diagrama de general de dicha plataforma.

Los componentes de la plataforma *Spartan 3A DSP 1800A* son:

- FPGA: Xilinx 3SD1800A – FG676 (1)
- Relojes (2):

[8] Diseño asistido por computador, Computer Assisted Design, es el uso de un amplio rango de herramientas computacionales que asisten a Ingenieros.

    - Oscilador de 125 MHz LVTTL SMT
    - Oscilador de 25.125 MHz LVTTL SMT
- Memorias (3):
    - 128 x 32-bit DDR2 SDRAM
    - 16M x 8 Paralela / BPI Configuration Flash
    - Configuración SPI 64Mb / Flash de Almacenamiento
- Interfaces (4):
    - 10 / 100 / 1000 PHY
    - Programación JTAG / Configuración de puertos
    - Puerto RS232
    - VGA de Bajo costo
- *Switches* y botones (5):
    - 8 LEDs de usuario
    - 8 DIP *Switch* para usuario
    - 4 Pulsadores para usuario
    - Pulsador de *Reset*
- I/O de usuario y expansión (6):
    - Digilent 6-pin *header* (2)
    - Conector de expansión EXP (2)
- Configuración y Depuración (7):
    - JTAG
    - Módulo de Conexión SystemACE^{TM}
    - Conector Eridon (SATA)

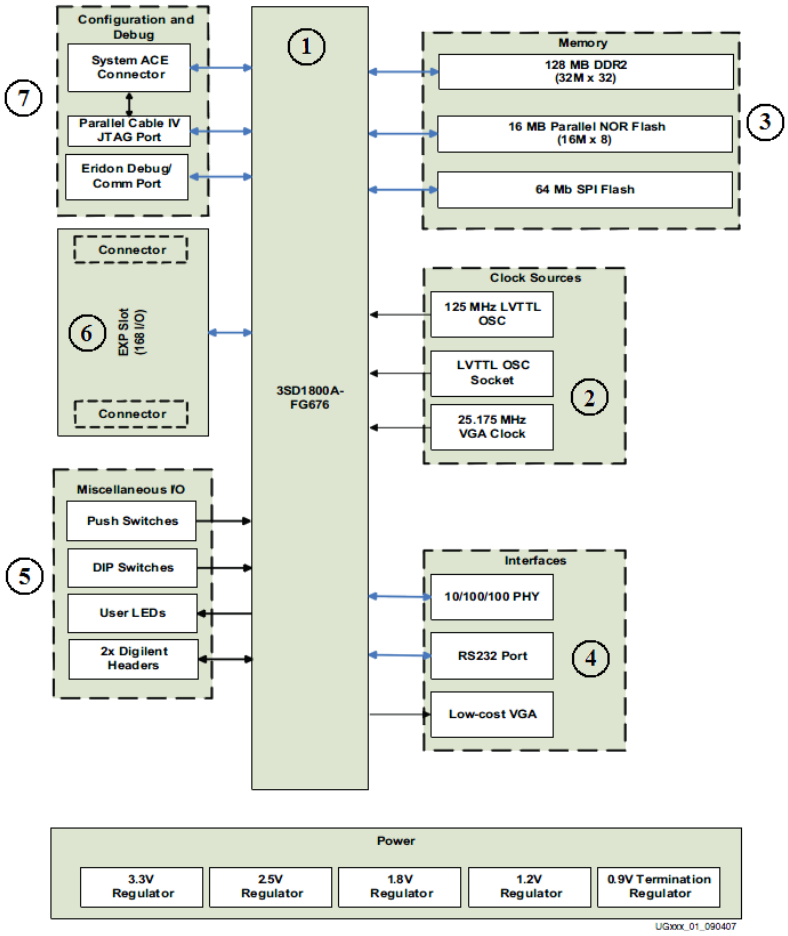

Figura 4: Diagrama de Bloques sistema de desarrollo Spartan 3A DSP [3]

Por medio del trabajo realizado con la plataforma *Spartan 3A DSP 1800A* de Xilinx se han encontrado maneras de desarrollar aplicaciones basadas en FPGAs (*Field Programmable Gate Arrays*) con gran flexibilidad y manejo de las mismas. La implementación de sistemas descritos en un lenguaje HDL en la plataforma es muy sencilla, por lo cual hoy en día este tipo de plataformas o sistemas de desarrollo son ampliamente utilizados. Esto se evidencia en el trabajo realizado en este proyecto.

1.4 Periféricos PMB (*Peripheral Module Bundle*) de Digilent

Los Periféricos PMB de Digilent están diseñados para el desarrollo de aplicaciones en los sistemas de desarrollo de Xilinx. Los Periféricos PMB se comunican con la plataforma *Spartan 3A DSP* a través de un conector de 6 que soporta señales de control digitales, incluyendo SPI y otros protocolos seriales. Los Periféricos PMB incluyen sensores, I/O, adquisición y conversión de datos, conectores, manejo de motores entre otros [4]. En la figura 5 se observan los 7 periféricos PMB de Digilent.

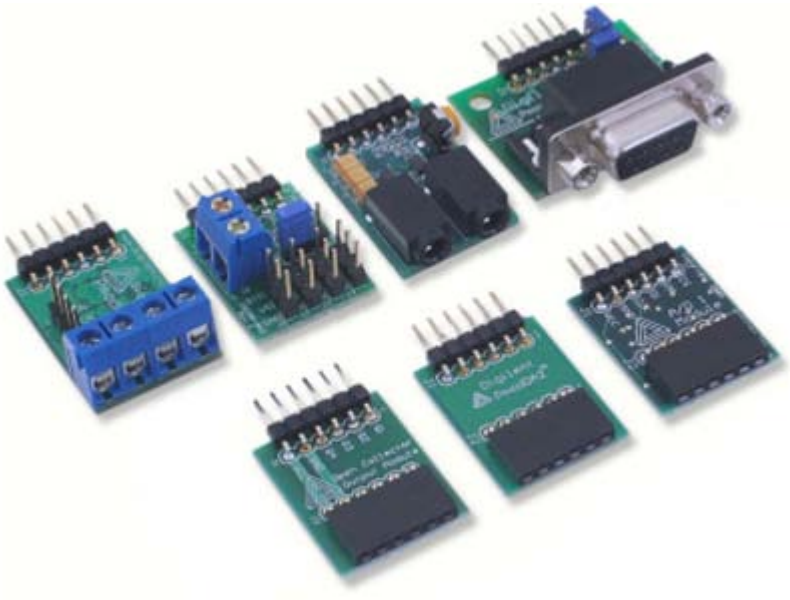

Figura 5: Periféricos PMB de Digilent [4]

Entre las ventajas de trabajar aplicaciones con los Periféricos PMB de Digilent se encuentra la facilidad de comunicación con la plataforma *Spartan 3A DSP* de Xilinx, la practicidad de los Periféricos, la aplicabilidad en los *embedded systems*.

Los Periféricos PMB a implementar en este proyecto son:

- PmodAD1: Conversor Analógico – digital de 12 bits
- PmodDA2: Conversor Digital – Analógico de 12 bits

- PmodOC1: Módulo de salida de colector abierto
- PmodHB3: Puente H
- PmodCON3: Control de 4 servo motores
- PmodRS232: Conector RS232
- PmodAMP1: Amplificador de Audio

Todos los periféricos PMB de Digilent se conectan a la alimentación del sistema de desarrollo *Spartan 3A DSP* de Xilinx. El conector de los periféricos PMB consta de 4 pines de comunicación con el sistema de desarrollo para el envío y la recepción de los datos y el control del Periférico; y consta de un pin conectado a la alimentación del sistema de desarrollo Vcc y un pin conectado a GND. Igualmente dependiendo de la aplicación el Periférico tendrá un conector J2 que permite su conexión a una respectiva aplicación: Motor DC, Servo Motores, Cable RS232, Señales analógicas, Señales digitales, dispositivos de alta corriente, altavoz y audífonos, este conector J2 dependerá del periférico PMB. En la figura 6 se muestra el esquema general de los periféricos PMB de Digilent.

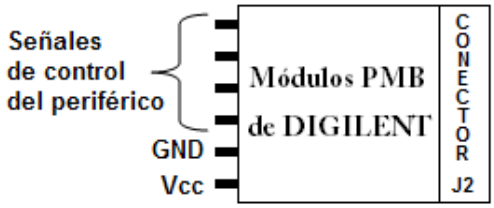

Figura 6: Esquema Periférico PMB

1.5 El software EDK (*Embedded Development Kit*) de Xilinx

Embedded Development Kit es un software CAD con una serie de herramientas de diseño que están basadas en un entorno de trabajo común, que permite el diseño

de un sistema de procesador empotrado completo para ser implementado en un FPGA (*Field Programmable Gate Array*) [5].

El EDK incluye:

- La interface con la plataforma de estudio de Xilinx (XPS).
- Una serie de herramientas para el diseño de *Embedded systems*.
- Procesadores IP para procesamiento embebido (*IP Cores*) y periféricos.
- Plataforma de estudio SDK (*Software Development Kit*), que puede ser usada para desarrollar aplicaciones software.

Para instalar el software EDK (*Embedded Development Kit*), es necesario también instalar el software ISE® (*Integrated Software Environment*), una herramienta de Xilinx para el desarrollo de sistemas digitales, requerido para implementar diseños en los dispositivos de lógica programable (PLDs) de Xilinx. El EDK depende de los componentes de ISE® para sintetizar el diseño hardware del microprocesador, mapear el diseño, generar y descargar el *bit stream*[9] [5]. Por otra parte el software EDK permite la creación, importación y adición al procesador IP de periféricos externos.

La plataforma de estudio de Xilinx (XPS) es una herramienta que permite diseñar un sistema de procesamiento embebido basado en el procesador MicroBlaze y el PowerPC, al cual se le pueden adicionar periféricos IP propios del sistema de desarrollo escogido o periféricos IP creados por el usuario.

También permite integrar el diseño hardware y el diseño software haciendo más práctico y flexible el desarrollo de aplicaciones embebidas.

[9] El término *bit stream* es frecuentemente utilizado para describir los datos de configuración a ser descargados en un FPGA.

Capítulo 2

2. Metodología de trabajo

2.1 Arquitectura general de los sistemas implementados

La arquitectura elegida en este proyecto consta de un procesador IP softcore, al cual se le adicionan módulos descritos en VHDL que controlan a los periféricos PMB de Digilent, todo esto implementado en una FPGA. Para este proyecto se utiliza un procesador IP softcore (MicroBlaze), una FPGA (Spartan 3A DSP 3SD1800A – FG676), todo esto implementado en el sistema de desarrollo *Spartan 3A DSP 1800A*.

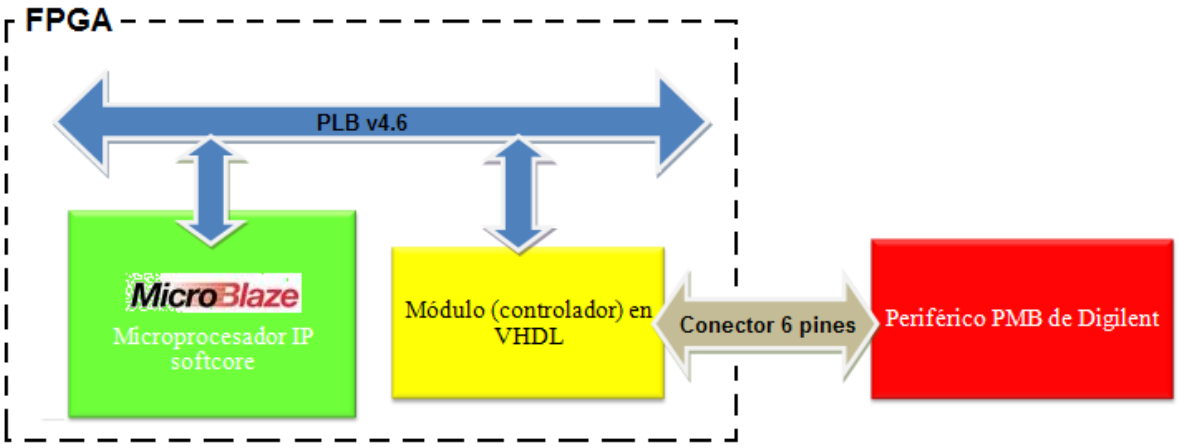

Figura 7. Arquitectura del proyecto.

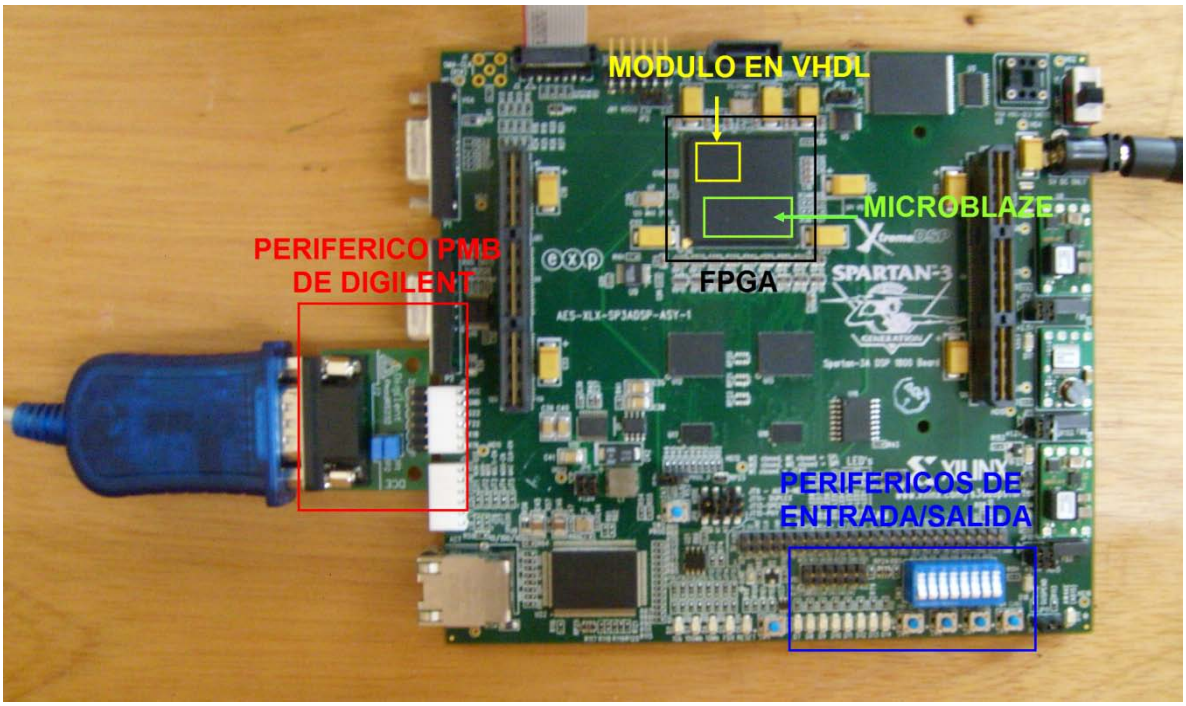

Figura 8: Enfoque de diseño general.

El direccionamiento del procesador MicroBlaze y la adición de los módulos descritos en VHDL para controlar los periféricos, se realizan por medio del software EDK (Embedded Development Kit) de Xilinx, específicamente a través de la herramienta XPS (Xilinx Platform Studio).

2.2 Flujo de trabajo

A la hora de diseñar cada una de las aplicaciones realizadas en este proyecto se siguió un flujo de trabajo que comienza con la descripción en VHDL de los módulos para controlar cada uno de los periféricos PMB, después se realizó la configuración general del microprocesador MicroBlaze para el sistema general, se adicionó cada módulo al MicroBlaze y finalmente se validó a través de una aplicación en software el correcto funcionamiento del módulo descrito. En la figura 8 se muestra el flujo de trabajo utilizado para el proyecto.

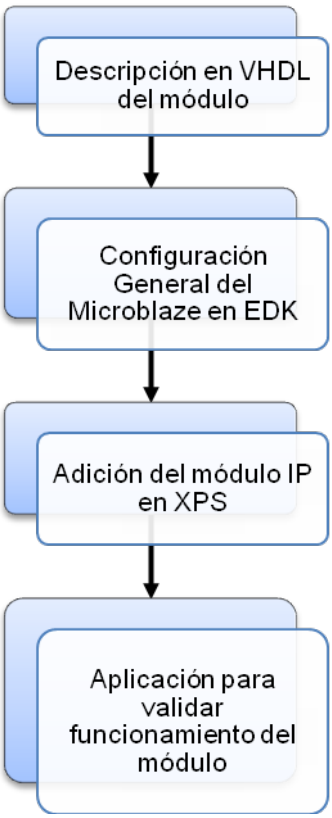

Figura 9: Flujo de trabajo del proyecto.

2.3 Descripción en VHDL del módulo para controlar el periférico:

Para la descripción de cada uno de los módulos que controlan los periféricos PMB se utilizó el lenguaje *Very High Description Language* (VHDL), siendo éste un lenguaje de sintaxis amplia y flexible que permite modelar y diseñar sistemas digitales con arquitectura funcional y secuencial.

Figura 10: Metodología de descripción.

Se utilizó una metodología de descripción en la que se define como primera etapa

el objetivo del módulo, como segunda etapa se realiza la descripción del módulo en VHDL y finalmente se hace una simulación de éste.

Esta metodología se presenta en la figura 10.

- Objetivo del módulo:

Se definió el objetivo del módulo a describir a través del estudio del funcionamiento del periférico PMB a controlar. Se identificaron las señales de entrada y salida necesarias para el módulo y se definieron las características generales y funcionales de cada periférico PMB.

- Descripción en lenguaje VHDL

Para la descripción del código en VHDL se utilizó la herramienta de diseño ISE 10.1 de Xilinx, que es un entorno de diseño integrado que consiste en un conjunto de herramientas para crear, simular e implementar diseños digitales en un dispositivo de lógica programable (FPGA, CPLD, etc).

Una vez descrito el código en VHDL se realiza una depuración del código que consiste una verificación de la sintaxis y la lógica de código en VHDL.

- Simulación del modulo en VHDL

Como etapa final de la descripción del módulo en VHDL se realizó una simulación con el fin de verificar el funcionamiento del módulo.

2.4 Configuración del MicroBlaze

Con el fin de configurar el sistema con el que se trabaja, se realizó la configuración del MicroBlaze, teniendo en cuenta el sistema de desarrollo utilizado, definiendo los parámetros del procesador como memoria, frecuencia de trabajo y Periféricos

IP de entrada/salida propios del sistema de desarrollo. En la figura 11 se presenta el esquema del sistema básico con el que se trabajó.

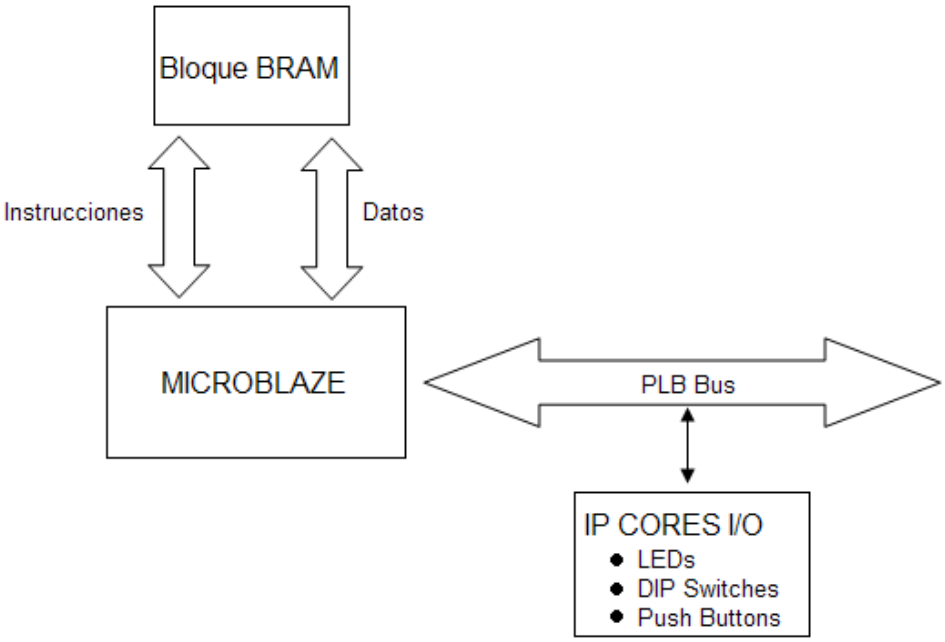

Figura 11. Esquema del sistema básico.

La configuración básica que se realizó es la siguiente:

PARAMETRO	**CONFIGURACION**
SISTEMA DE DESARROLLO	Xilinx Spartan 3A DSP 1800ª Starter Board 1
PROCESADOR	MicroBlaze
FRECUENCIA RELOJ DE REFERENCIA *Frecuencia del bus del procesador*	125 MHz 62.5 MHz
CONFIGURACION PROCESADOR *Debug I/F* *Memoria Local (BRAM)*	 *On chip H/W debug module* *8 KB*

Memoria cache	*Deshabilitada*
DISPOSITIVOS DE ENTRADA/SALIDA INTERNOS	LEDs_8Bit Periférico: GPIO[10] Push_Buttons Periférico: GPIO DIP_Switches_8Bit Periférico: GPIO

Tabla 1. Configuración inicial del sistema (Procesador y Periféricos)

El resultado de la configuración básica del sistema creado se muestra en la figura 12.

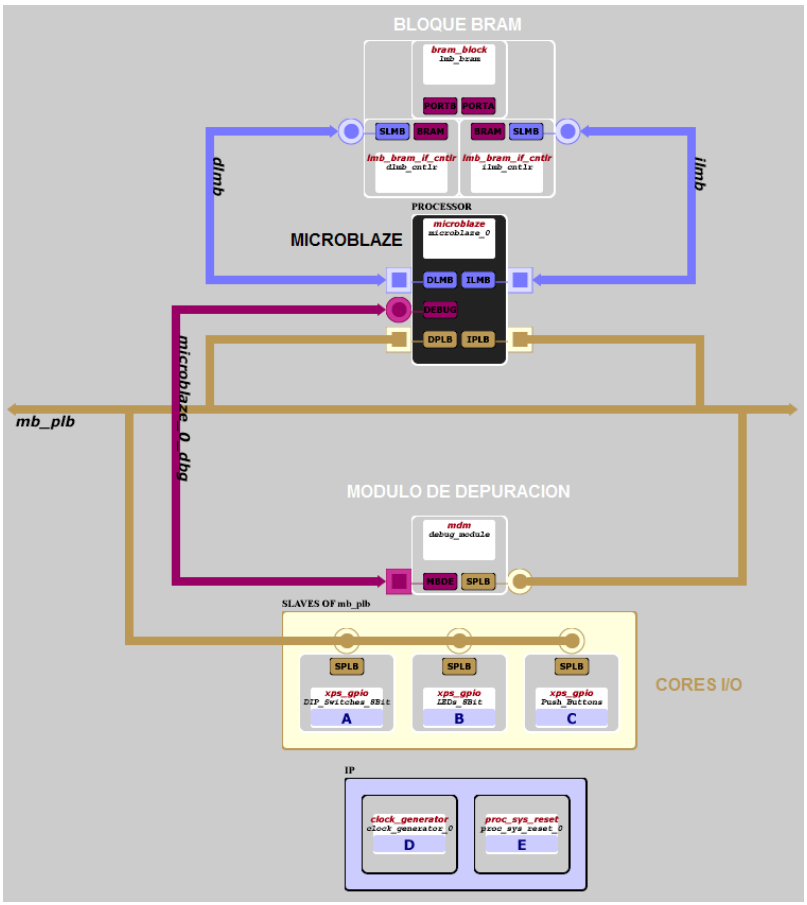

Figura 12. Diagrama de bloques generado por XPS para el sistema básico.

[10] GPIO (General Purpose I/O): Periférico de propósito general de entrada/salida.

2.5 Adición del módulo descrito en VHDL al microprocesador embebido MicroBlaze

El módulo descrito en VHDL se adicionó al MicroBlaze por medio de las herramientas que ofrece XPS[11]. Para su adición se creó un nuevo periférico IP para incluir en él el módulo descrito en VHDL. El esquema general con la adición del módulo descrito en VHDL se presenta en la figura 13.

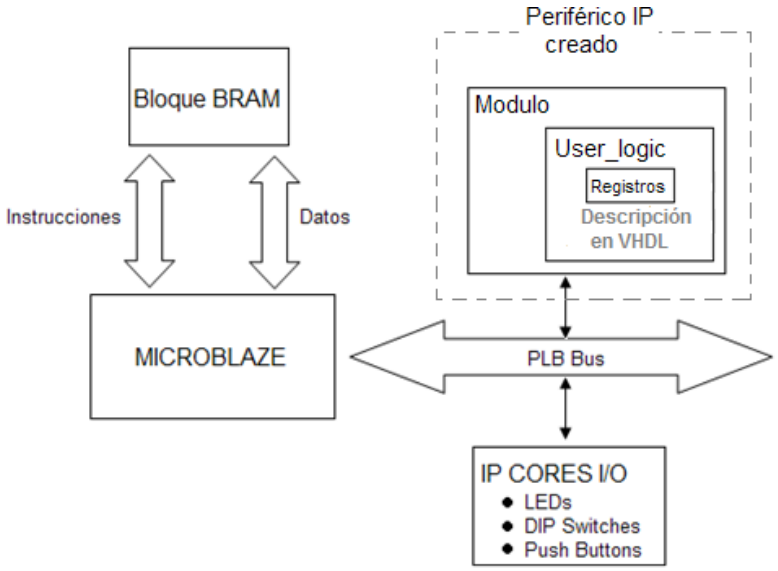

Figura 13. Esquema del sistema con el periférico IP creado y adherido al MicroBlaze.

En la creación del periférico IP el software XPS genera dos archivos: modulo.vhd y user_logic.vhd; estos archivos son generados en: *<nombre_del_proyecto>\pcores\<modulo_v1_00_a>\hdl\vhdl*.

En la figura 14 se muestra el esquema del periférico IP creado. El periférico IP se conecta al bus local del procesador, donde están conectados también los periféricos IP de entrada/salida.

[11] XPS: Xilinx Platform Studio

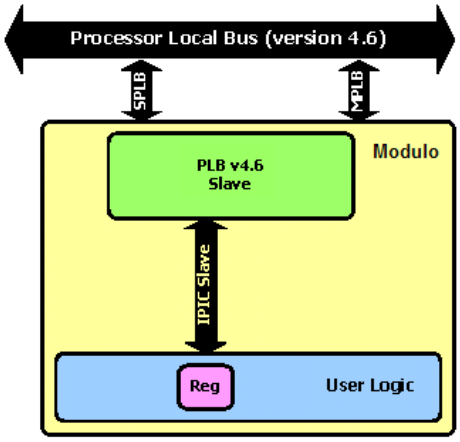

Figura 14. Esquema del periférico IP creado.

El periférico IP permite el acceso de datos desde software a través de registros esclavos internos (Figura 15). Estos registros son de 32 bits, el usuario elige la cantidad de registros que desea utilizar y son creados en el archivo “User Logic” donde se implementó el módulo en VHDL diseñado.

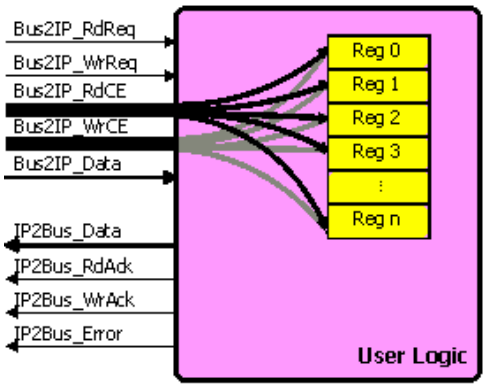

Figura 15. Registros esclavos internos creados para acceso desde software.

Finalmente se adicionó el módulo descrito en VHDL en el archivo “User Logic”, teniendo en cuenta los registros esclavos internos para la comunicación con el software, se crearon las señales externas al periférico modificando el archivo modulo.vhd de más alto nivel.

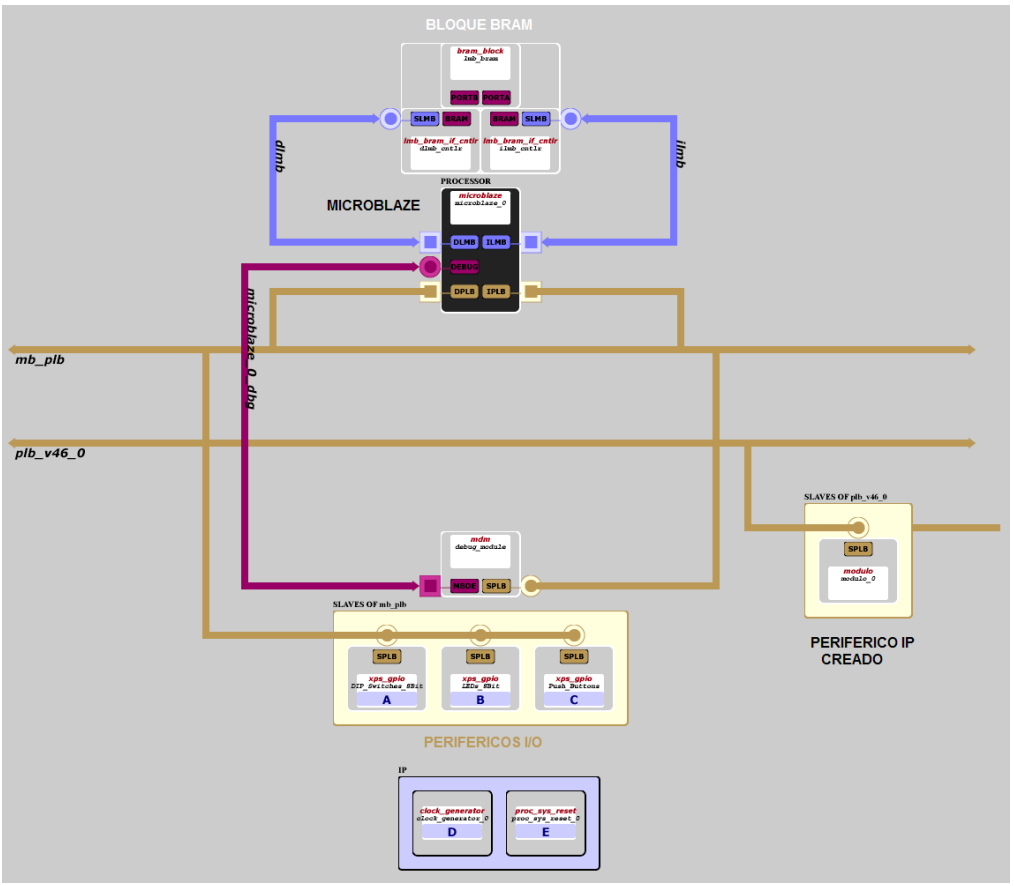

Figura 16. Diagrama de Bloques del sistema con el periférico IP creado.

2.6 Aplicación para validar funcionamiento del módulo descrito en VHDL

Para la validación de cada uno de los módulos (controladores) descritos en VHDL se realizó una aplicación en lenguaje C a través de las herramientas de software que tiene EDK. Como primer paso en el desarrollo de la aplicación se realizó una descripción general de la aplicación teniendo claro el funcionamiento de cada uno de los periféricos PMB, luego se realizó un diagrama de flujo de la aplicación para simplificar el paso al lenguaje C, se generaron las librerías y drivers en el software EDK necesarios para poder hacer la descripción de la aplicación en lenguaje C, luego se compiló el software realizado para cada módulo, se descarga el *bit stream* que contiene la información del hardware a implementar (*.bit) y software

(*.elf) del sistema y finalmente se pone en funcionamiento el sistema con la aplicación para validar su correcto funcionamiento. El flujo de diseño se presenta en la figura 17.

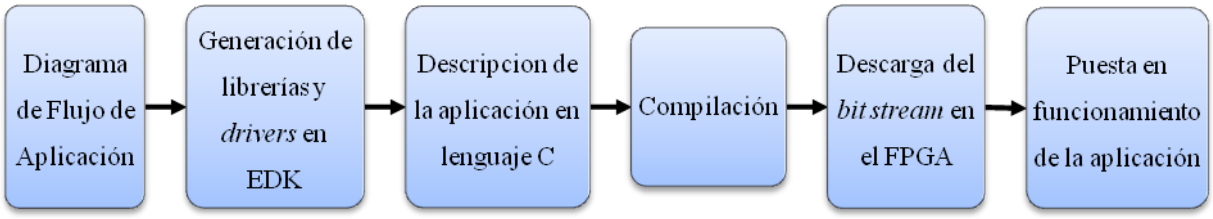

Figura 17: Flujo de diseño para validar funcionamiento de cada módulo descrito en VHDL.

- *Diagrama de Flujo:*

Para cada uno de los módulos se pensó en una aplicación sencilla que permitiera cumplir con el objetivo de validar el buen funcionamiento de cada uno de los módulos descritos en VHDL para controlar los periféricos PMB. Esta aplicación también es realizada para comprobar que realmente el módulo se encuentra adicionado y conectado al microprocesador MicroBlaze. El Microblaze es el que se encarga de hacer el procesamiento de la aplicación.

- *Generación de librerías y drivers:*

Una vez adicionado el módulo en VHDL al microprocesador MicroBlaze y después de haber generado el archivo .bit, se generan las librerías y los *drivers* para poder realizar la aplicación en C. En la figura 18 se muestra las etapas en la generación de las librerías y *drivers*, esto se hace a través de la herramienta LibGen del entorno XPS donde trabajamos.

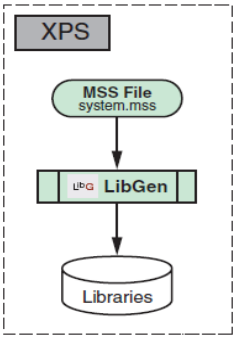

Figura 18: Etapas en la generación de las librerías y drivers [5].

Figura 19: Generación de librerías y Drivers en EDK.

Al generar las librerías y los *drivers* se genera el archivo xparameters.h. Este archivo se debe incluir en el código en C.

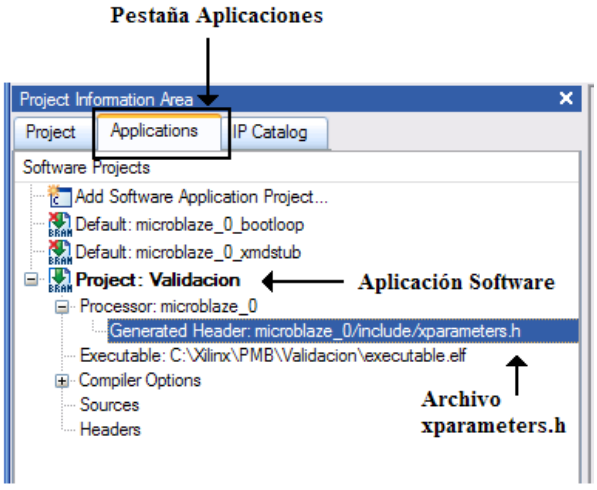

Figura 20: Pestaña de aplicaciones en el área de información del proyecto.

- *Descripción en lenguaje C:*

Se realizó la descripción en lenguaje C de la aplicación de cada módulo. Se tuvieron en cuenta las librerías propias de cada uno de los periféricos IP utilizados y sus funciones. Se incluyeron las librerías en la cabecera de la aplicación como se muestra en la figura 21.

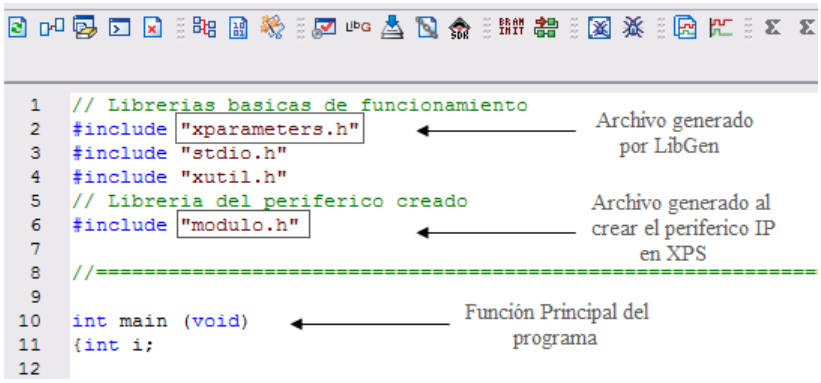

Figura 21: Librerías básicas de Funcionamiento utilizadas.

El software EDK crea librerías y funciones propias de cada módulo creado, para nuestro caso todos los módulos creados trabajan con registros esclavos internos para el manejo de los datos de entrada y salida hacia el procesador. Estos registros esclavos internos son accedidos desde software a través de dos funciones que son creadas por EDK, estas dos funciones son:

void MODULO_mWriteSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset, Xuint32 Value)

Xuint32 MODULO_mReadSlaveRegn(Xuint32 BaseAddress, unsigned RegOffset)

Donde *BaseAddress* es la dirección base del módulo creado, *RegOffset* es el *offset* del registro esclavo n para escribir o para leer y *Value* es el valor que se desea escribir en el registro n. La segunda función retorna el valor leído del registro n.

Estas funciones se encuentran en:

<nombre_del_proyecto>\drivers\<modulo_v1_00_a>\src

- *Compilación:*

La compilación la realiza EDK generando un archivo .elf[12]. En la figura 22 se muestra como se realiza la compilación y en la figura 23 se muestra el archivo generado de ésta compilación.

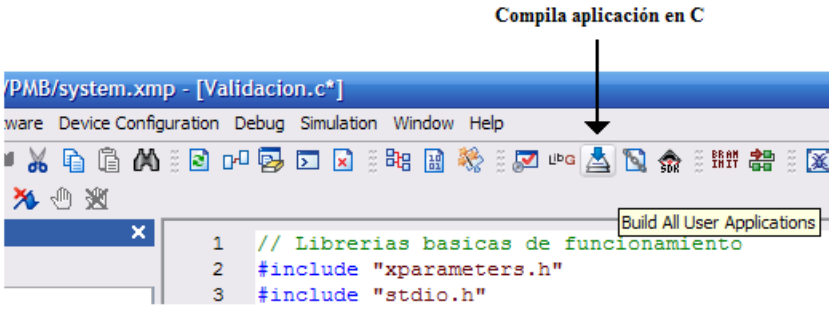

Figura 22: Compilación de aplicación en C.

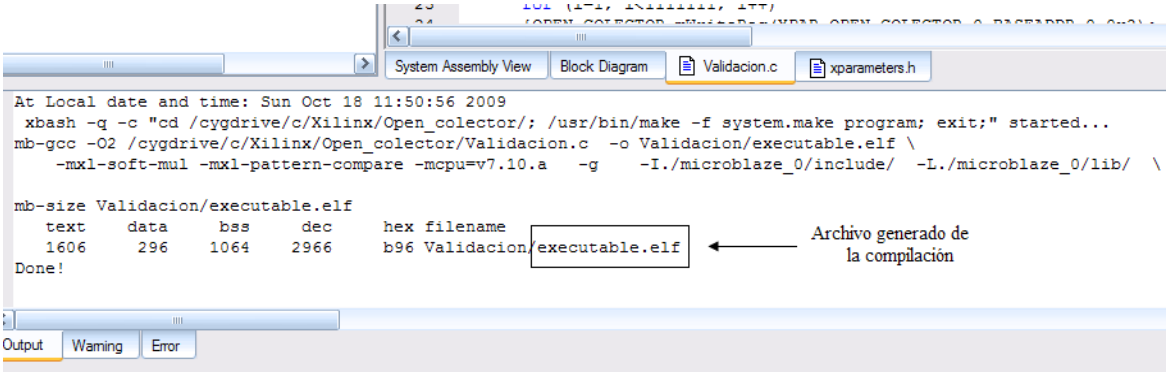

Figura 23: Archivo generado de la compilación de la aplicación.

- *Descarga del bit stream:*

Una vez compilada la aplicación en C se generó un archivo .bit[13] que contiene el hardware (*.bit) y el software (*.elf), esto para descargarlo en la *spartan 3A DSP 1800A* (FPGA) de Xilinx. En la figura 24 se muestra como se realiza la unión de la

[12] Archivo ejecutable creado en la compilación.
[13] Archivo que contiene el bit stream para ser descargado en la FPGA

parte hardware y software en un solo archivo y la descarga de éste archivo en la FPGA.

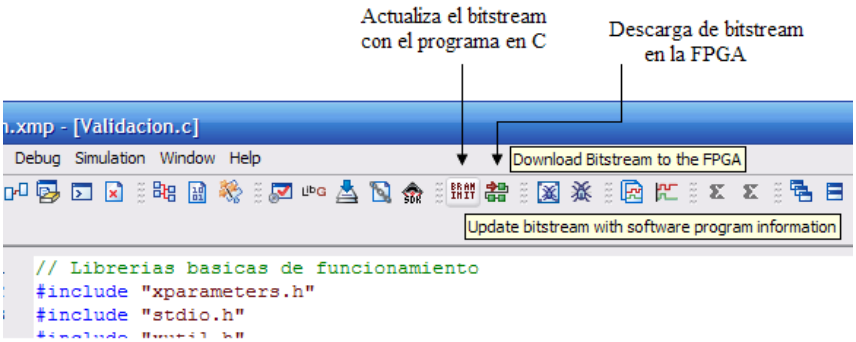

Figura 24: Descarga de bit stream en la FPGA.

- *Puesta en funcionamiento de la aplicación:*

Finalmente descargado el sistema en la FPGA se puso en funcionamiento el sistema con la aplicación realizada y se comprobó su funcionamiento para hacer los ajustes necesarios de hardware y software si es el caso. En la figura 25 se muestra una foto tomada en el laboratorio para la aplicación realizada para uno de los módulos descritos del proyecto.

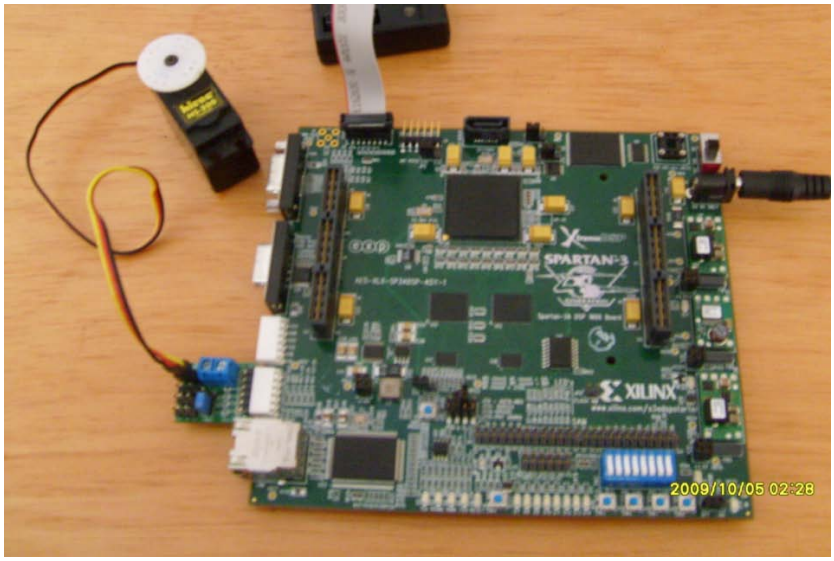

Figura 25: Imagen tomada en el laboratorio para uno de los Periféricos del proyecto.

Capítulo 3

3. Módulos PMB

3.1 Módulo: Conversor Analógico Digital (PmodAD1™)

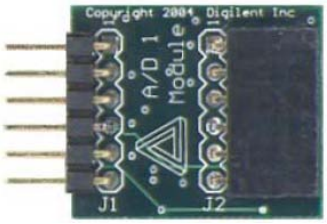

Figura 26: Periférico PmodAD1™

INTRODUCCION:

El periférico PmodAD1™ de Digilent convierte una señal análoga de entrada de rango entre 0 y 3.3 V en un valor digital de 12 bits con un rango entre 0 y 4095. Este periférico contiene dos canales simultáneos de conversión, cada uno cuenta con un conversor de 12 bits, de referencia ADCS7476MSPS, y un filtro Sallen Key anti aliasing[14] de dos polos. El filtro limita el ancho de banda de la señal de entrada análoga a un rango de frecuencia adecuado con la velocidad de muestreo del conversor.

[14] Filtro que atenúa el contenido armónico superior a la frecuencia de Nyquist

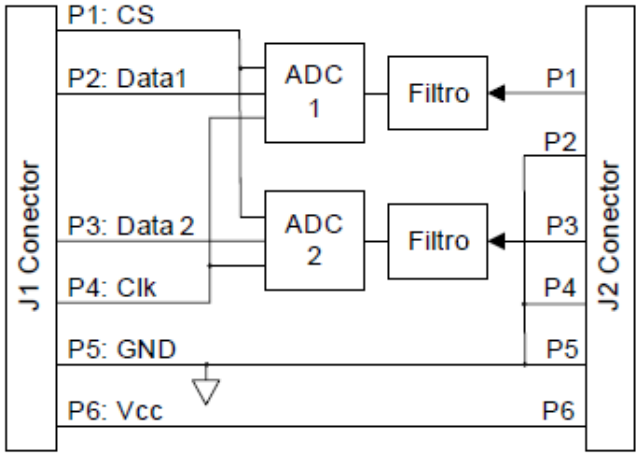

Figura 27: Diagrama del PmodAD1™

El periférico PmodAD1™ tiene como entradas dos señales análogas que pasan por los filtros y luego llegan a los conversores. Los conversores convierten estas señales y envían a la salida un valor digital de 12 bits de forma serial. Estas son las señales Data1 y Data2 que se observan en la figura 20. El conversor envía el valor convertido a través de un bus estándar serial controlado por una señal de reloj conectada al periférico. Igualmente esta señal de reloj controla la conversión. La máxima frecuencia a la que trabaja el bus serial es de 20 MHz.

La señal CS es la que permite iniciar el proceso de conversión de datos. El proceso de conversión comienza en el flanco de bajada de la señal CS.

Por otro lado la salida digital de los datos es serial, y el conversor envía primero tres ceros y luego comienza a enviar los datos partiendo del MSB[15].

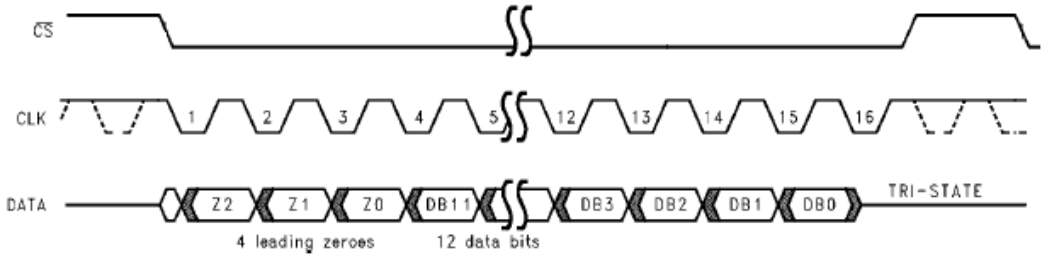

Figura 28: Diagrama de tiempo conversor ADCS7476 [11].

[15] Bit más significativo, Most Significant Bit, en sus siglas en inglés, es el bit, que de acuerdo a su posición, tiene el mayor valor

DESCRIPCION EN VHDL:

La descripción en VHDL del módulo que controla el periférico PmodAD1™ de Digilent se encuentra en el anexo A.

APLICACIÓN EN SOFTWARE:

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodAD1™ se realizó una aplicación que consiste en suministrar un valor de voltaje entre 0 y 3.3 V al periférico PmodAD1™, el procesador recibe el valor convertido de 12 bits que viene del controlador diseñado y lo compara con un rango de valores permitidos, para ésta aplicación se escogió un rango entre 1 y 2.6 V. Si el valor se encuentra dentro del rango, el procesador activa todos los LEDs[16] del sistema de desarrollo, por el contrario si está fuera del rango, los LEDs se mantendrán apagados.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodAD1™ se encuentra en el anexo A.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

3.2 Módulo: Conversor Digital Analógico (PmodDA2™)

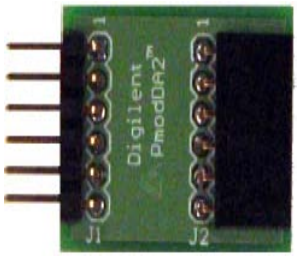

Figura 29: Periférico PmodDA2™

[16] Light Emitter Diode: Diodo emisor de Luz.

INTRODUCCION:

El Periférico PmodDA2™ de Digilent convierte una señal digital de entrada de 12 bits a una señal de tensión analógica de rango entre 0 y 3.3 V. Este Periférico contiene dos canales simultáneos de conversión, cada uno cuenta con un conversor digital – analógico de 12 bits, de referencia DAC121S101.

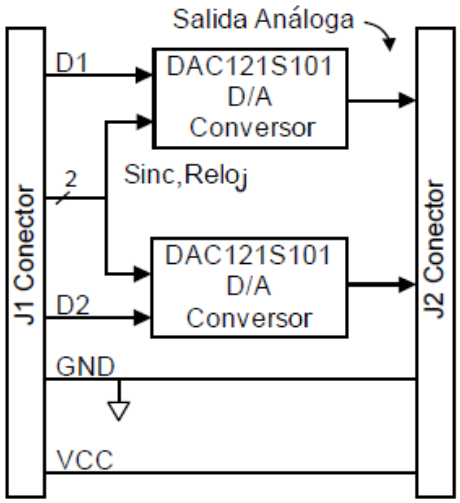

Figura 30: Diagrama del PmodAD2™

El modulo PmodDA2™ tiene como entrada dos señales digitales (D1, D2) que son recibidas por los conversores A/D de 12 bits de forma serial SPI (ver figura 30). El conversor recibe el valor digital a través de un bus estándar serial controlado por una señal de reloj conectada al Periférico. Igualmente esta señal de reloj controla la conversión.

La señal SYNC es la que permite iniciar el proceso de conversión de datos. El proceso de conversión comienza en el flanco de bajada de la señal SYNC.

Por otro lado la entrada al conversor D/A debe ser serial, es decir bit a bit partiendo del MSB (ver figura 31).

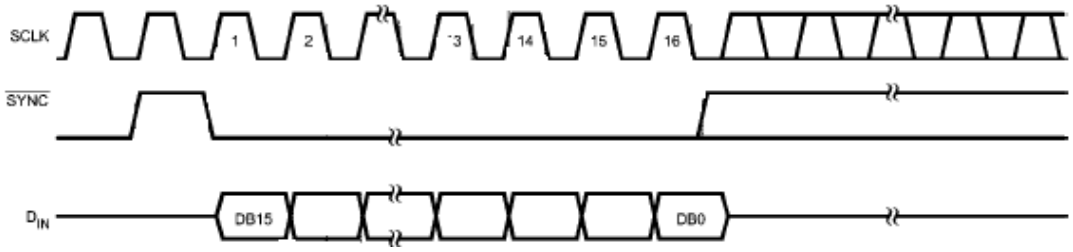

Figura 31: Diagrama de tiempo conversor DAC081S101 [12].

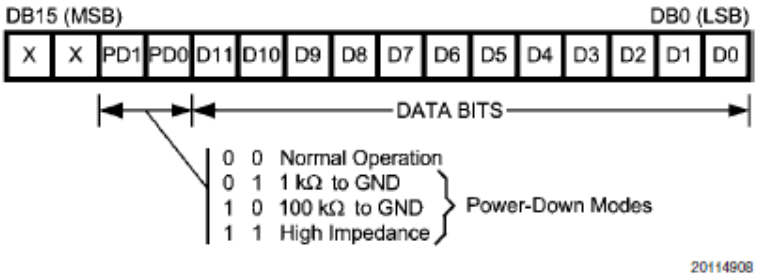

Figura 32: Configuración registro de entrada para DAC081S101 [12].

DESCRIPCION EN VHDL:

La descripción en VHDL del módulo que controla el periférico PmodDA2^{TM} de Digilent se encuentra en el anexo B.

APLICACIÓN EN SOFTWARE

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodDA2^{TM} se realizó una aplicación que consiste en enviar una secuencia de valores digitales al periférico PmodDA2^{TM} con un tiempo establecido de mantenencia y observar la salida analógica. Todo esto se programó en lenguaje C utilizando las funciones propias y las funciones especiales que se generaron con el módulo creado en VHDL cuando se añadió éste al procesador.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodDA2^{TM} se encuentra en el anexo B.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

3.3 Módulo: Salida Colector Abierto (PmodOC1™)

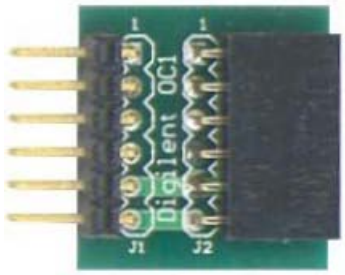

Figura 33: Periférico PmodOC1™

INTRODUCCION:

El periférico PmodOC1™ de Digilent permite manejar dispositivos de alta corriente usando transistores de salida MMBT3904. Los transistores son controlados por señales lógicas que vienen del sistema de desarrollo y para nuestro caso son enviadas desde el programa del procesador.

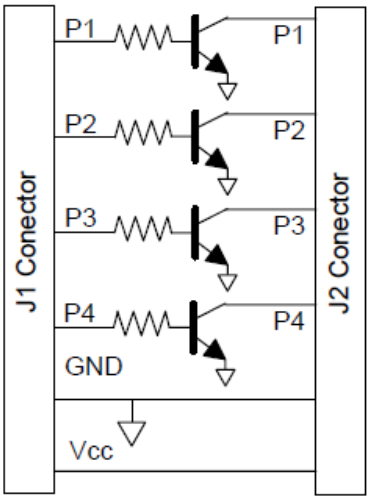

Figura 34: Diagrama del PmodOC1™

El periférico PmodOC1™ contiene 4 transistores MMBT3904 [13] de 100 mA (200 mA Máximo) con un voltaje de máximo entre colector-emisor de 40V. Cada transistor funciona independientemente de los demás, luego estos pueden trabajar individualmente o simultáneamente. Cada uno de los transistores tiene un colector que conduce corriente entre un dispositivo externo conectado en J2 (ver figura 34) y la conexión de tierra establecida por el periférico. Cuando el transistor se enciende a través de un nivel lógico alto que viene de J1 (ver figura 34), la corriente fluye a través del circuito conectado en el pin correspondiente en J2. El periférico PmodOC1™ tiene 4 diodos de protección en la salida que evitan que se produzcan daños en los transistores, esto debido a cargas inductivas.

DESCRIPCION EN VHDL:

La descripción en VHDL del módulo que controla el periférico PmodOC1™ de Digilent se encuentra en el anexo C.

APLICACIÓN EN SOFTWARE:

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodOC1™ se realizó una aplicación que consiste en enviar una secuencia de encendido para cuatro LEDs conectados a las cuatro salidas de colector abierto. Todo esto se programó en lenguaje C utilizando las funciones propias y las funciones especiales que se generaron con el módulo creado en VHDL cuando se añadió éste al procesador.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodOC1™ se encuentra en el anexo C.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

3.4 Módulo: Puente H (PmodHB3™)

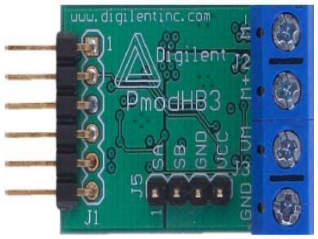

Figura 35: Periférico PmodHB3™

INTRODUCCION:

El periférico PmodHB3™ de Digilent permite trabajar aplicaciones donde se utilizan señales de nivel lógico, para impulsar pequeños y medianos motores. Este periférico puede controlar la dirección de giro de un motor DC, igualmente su velocidad.

Este periférico cuenta con una entrada de alimentación de 12V, conectada al bloque J3, una señal de control de dirección (DIR) de nivel lógico, la cual está encargada del cambio del sentido de giro del motor, dos señales (SA, SB) que brindan una retroalimentación de la velocidad del motor al sistema, una señal de control de velocidad (EN) que determina la regulación del cambio de la velocidad del motor, es controlada por modulación por ancho de pulsos (PWM).

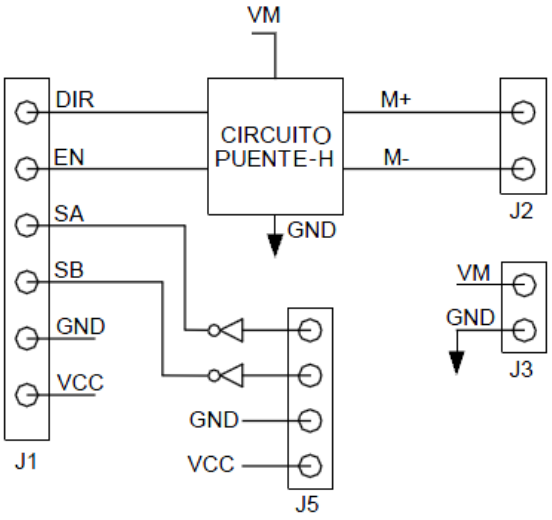

Figura 36: Diagrama del PmodHB3™

La modulación por ancho de pulsos (PWM) es una técnica utilizada para controlar dispositivos donde se modifica el ciclo de trabajo de una señal periódica de entrada.

En la figura 37 se ilustra un sistema PWM (*Pulse Width modulation*) con frecuencia de entrada de 2 KHz. La velocidad del motor es controlada ajustando el tiempo en el que el pulso se mantiene en nivel alto. En la figura 37 se muestra una señal con ciclo de trabajo del 10%, 50% y 75%. Este cambio en el ciclo de trabajo de la señal permite que varíe la velocidad del motor, ya que se tiene como referencia la velocidad máxima del motor que seria para un ciclo de trabajo del 100%.

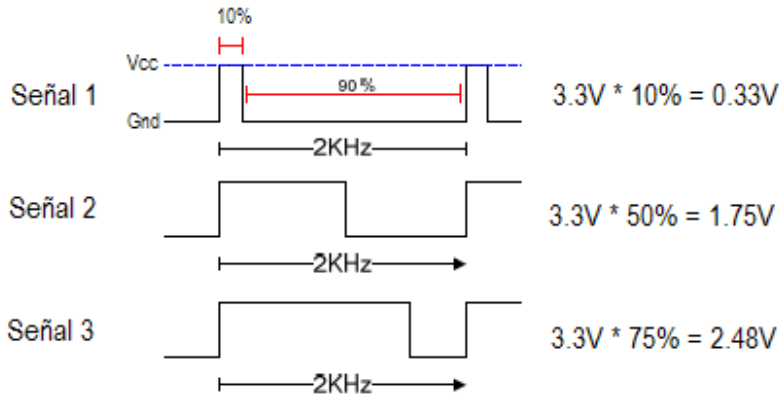

Figura 37: Sistema PWM.

DESCRIPCION EN VHDL:

La descripción en VHDL del módulo que controla el periférico PmodHB3™ de Digilent se encuentra en el anexo D.

APLICACIÓN EN SOFTWARE:

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodHB3™ se realizó una aplicación que consiste en mover un motor DC en una dirección con una velocidad determinada, luego pararlo, cambiarle el sentido de giro y la velocidad y finalmente volver a pararlo y cambiarle

de nuevo el sentido de giro y la velocidad. Todo esto se programó en lenguaje C utilizando las funciones propias y las funciones especiales que se generaron con el módulo creado en VHDL cuando se añadió éste al procesador.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodHB3™ se encuentra en el anexo D.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

3.5 Módulo: Control de servo motor (PmodCON3™)

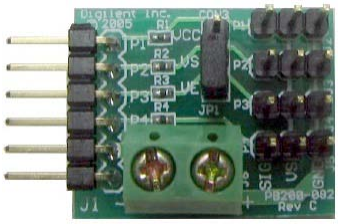

Figura 38: Periférico PmodCON3™

INTRODUCCION:

El periférico PmodCON3™ de Digilent permite el manejo de cuatro servo motores para aplicaciones de aeromodelismo y carros de radio control. Los servo motores son usados en un sin número de aplicaciones robóticas y están diseñados para moverse a una posición deseada y fijarse en ésta.

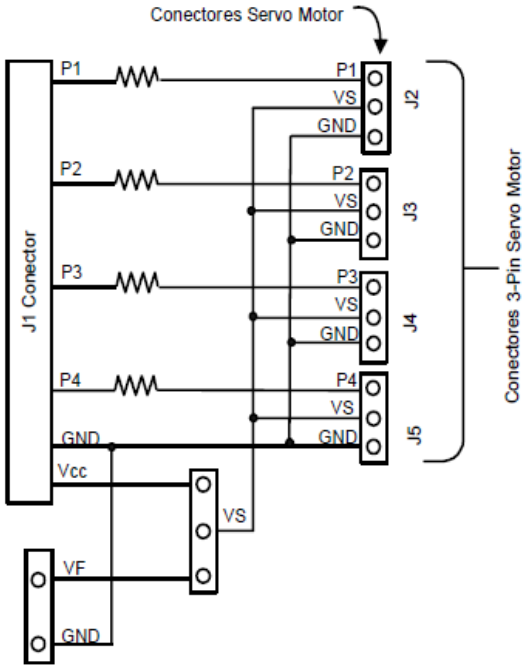

Figura 39: Diagrama del PmodCON3^{TM}

El periférico PmodCON3^{TM} tiene como entradas cuatro conectores de 3 pines donde van conectados los cuatro servo motores. Cada conector tiene una señal de control que es por donde se envía la señal PWM al servo motor, tiene la señal de alimentación y la señal de tierra. El periférico permite trabajar con una alimentación interna del sistema de desarrollo (para aplicaciones de baja potencia) o también utilizar una alimentación externa (para aplicaciones de alta potencia. A través de un jumper se elige la fuente de alimentación con la que se desea alimentar a los servos motores.

Para el control de los servos se utiliza una señal PWM (*Pulse Width modulation*) o de modulación por ancho de pulsos, la cual controla la dirección y el ángulo de rotación del servo motor.

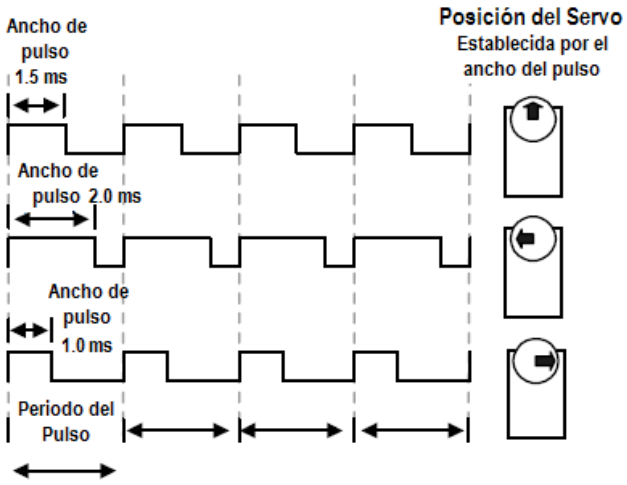

Figura 40: Diagrama de Control del servo

DESCRIPCION EN VHDL:

La descripción en VHDL del módulo que controla el periférico PmodCON3^{TM} de Digilent se encuentra en el anexo E.

APLICACIÓN EN SOFTWARE:

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodCON3^{TM} se realizó una aplicación que consiste en enviar una secuencia de valores al PWM del controlador diseñado en VHDL para realizar un control en la dirección y el ángulo del servo motor. Todo esto se programó en lenguaje C utilizando las funciones propias y las funciones especiales que se generaron con el módulo creado en VHDL cuando se añadió éste al procesador.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodCON3^{TM} se encuentra en el anexo E.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

3.6 Módulo: PmodRS232™

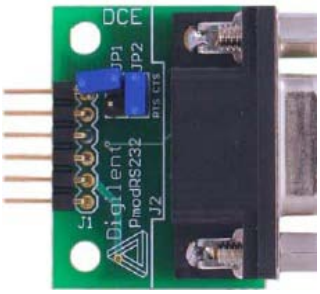

Figura 41: Periférico PmodRS232™

INTRODUCCION:

El periférico PmodRS232^{TM} de Digilent permite la comunicación serial de datos utilizando el protocolo RS232. Este periférico convierte voltajes de niveles lógicos usados por el sistema de desarrollo a voltajes usados en la comunicación serial RS232. Los niveles a los que trabaja el periférico PmodRS232^{TM} son digitales de +3 a 12 V (0 lógico) y -3 a 12 V (1 lógico), para la entrada y salida de datos.

Las características del periférico PmodRS232^{TM} son: un circuito integrado Max3232, un conector DB9 y un conector de 6 pines, señales de transmisión y recepción, una señal CTS (Clear to Send) que avisa cuando el sistema está listo para enviar datos y una señal RTS (Request to Send) que avisa la solicitud para envío de datos, es decir que está listo para recibir datos.

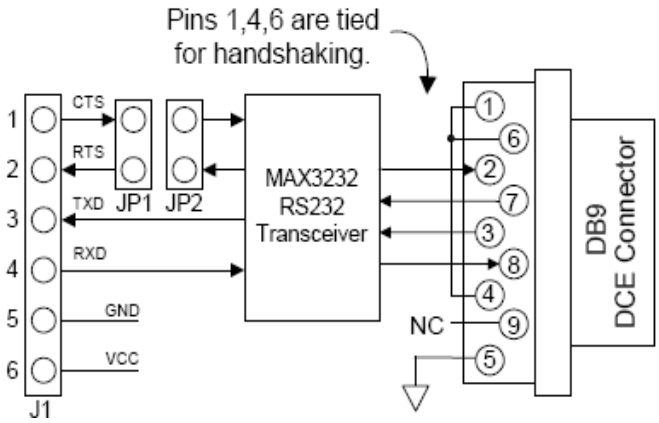

Figura 42: Diagrama del PmodRS232™

El periférico PmodRS232™ se puede configurar como un puerto serial de 3 conexiones (Una señal transmite, otra señal recibe y una conexión a tierra), o como un puerto serial de 5 conexiones con dos señales adicionales: RST y CTS (señales de *handshaking*[17]).

La señal CTS (libre para envío) se puede conectar a la entrada de un transmisor y la señal RST (solicitud de envío) se puede conectar a la salida de un receptor. Estas conexiones se realizan en los bloques JP1 y JP2 tal como se puede observar en la figura 42.

Los bloques JP1 y JP2 se utilizan para configurar el periférico PmodRS232™ bien sea como un puerto de 3 o 5 señales. El pin 1 y 2 del bloque JP1 se conecta a los pines 1 y 2 del conector J1 respectivamente. Los pines 1 y 2 del bloque JP2 se conectan a la señal CTS transmisor y la señal RST receptor respectivamente.

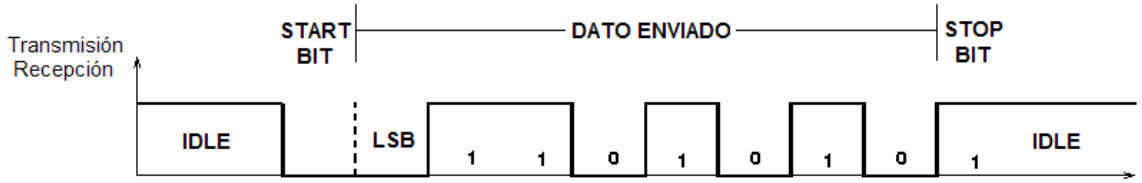

Figura 43: Transmisión y recepción RS232.

[17] Protocolo utilizado para el control de flujo desde hardware.

En la figura 43 se muestra el protocolo serial de transmisión y recepción utilizado para el módulo descrito en VHDL. Este protocolo es controlado por el módulo UART que permite enviar y recibir datos de manera asíncrona.

DESCRIPCION EN VHDL:

La descripción en VHDL del Periférico que controla el periférico PmodRS232^{TM} de Digilent se encuentra en el anexo F.

APLICACIÓN EN SOFTWARE:

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodRS232^{TM} se realizó una aplicación que consiste en enviar un dato desde software a MATLAB[18] y realizar una operación con el dato luego enviarlo de vuelta al MicroBlaze y mostrar el dato en los LEDs del sistema de desarrollo. Todo esto se programó en lenguaje C utilizando las funciones propias y las funciones especiales que se generaron con el módulo creado en VHDL cuando se añadió éste al procesador.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodRS232^{TM} se encuentra en el anexo F.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

[18] MATLAB (abreviatura de *MATrix LABoratory*, "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (lenguaje M).

3.7 Módulo: Amplificador de Audio (PmodAMP1™)

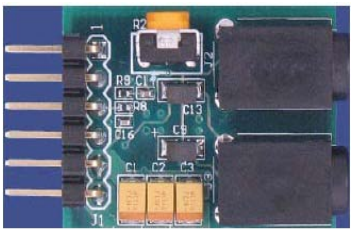

Figura 44: Periférico PmodAMP1™

INTRODUCCION:

El Periférico PmodAMP1^{TM} Amplificador altavoz/audífono de Digilent amplifica señales de audio de baja potencia para auricular estéreo o para altavoz monofónico. El altavoz utiliza la señal izquierda de la entrada estéreo.

La entrada de audio para el Periférico PmodAMP1^{TM} es J1 (conector de 6 pines). Un *Jack* de audio estéreo (1/8 de pulgada) se utiliza como salida para el auricular y un *Jack* de audio monofónico (1/8 de pulgada) se utiliza para la salida del altavoz.

El Periférico PmodAMP1^{TM} cuenta con un amplificador de audio LM4838 y dos filtros pasa bandas en las entradas del amplificador. Estos filtros pasa bandas en la entrada tienen una frecuencia pasa altos de corte aproximada de 150 Hz y una frecuencia pasa bajos de corte aproximadamente de 8 kHz.

La señal de entrada para el Periférico PmodAMP1^{TM} puede ser analógica, con rango entre 0 y Vcc o puede ser digital típicamente una señal PWM (Modulación por ancho de pulsos).

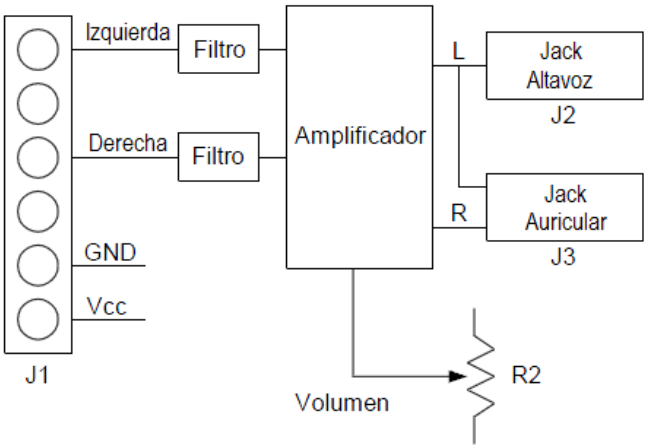

Figura 45: Diagrama de bloques del PmodAMP1™

El filtro paso bajos en la entrada actúa como un filtro de reconstrucción para convertir la señal digital PWM en un voltaje analógico en la entrada del amplificador.

El voltaje de entrada, cualquiera sea su naturaleza (analógica o digital), es filtrado por el filtro pasa bandas, amplificado y enviado a los *Jack* de salida de audio, tanto al del altavoz como al auricular. J2 es el conector de la salida de altavoz, J3 es el conector de la salida de auriculares. Tanto el auricular como el altavoz pueden ser conectados simultáneamente. El potenciómetro R2, es un control de volumen y se utiliza para ajustar el nivel de salida.

DESCRIPCION EN VHDL:

La descripción en VHDL del módulo que controla el periférico PmodAMP1^{TM} de Digilent se encuentra en el anexo G.

APLICACIÓN EN SOFTWARE:

Para validar el buen funcionamiento del módulo diseñado en VHDL para el control del periférico PmodAMP1^{TM} se realizó una aplicación que consiste en generar una

escala musical natural tanto en el altavoz como en los audífonos. Para generar la escala se calculan las frecuencias para generar las notas naturales de la escala de Do Mayor (Do, Re, Mi, Fa, Sol, La, Si y Do). El controlador genera una señal cuadrada a la frecuencia deseada, luego lo que se busca con la aplicación es comprobar que el amplificador de audio realmente funciona más que el mismo controlador, ya que este solo genera una señal a la frecuencia especificada en el programa realizado en C para la aplicación. Se utilizan las funciones propias y las funciones especiales que se generan cuando se crea el controlador en el software EDK y se añade éste al procesador.

El código en lenguaje C de la aplicación para el módulo descrito en VHDL que controla el periférico PmodAMP1^{TM} se encuentra en el anexo G.

Finalmente se comprobó que el módulo descrito en VHDL funciona correctamente, ya que con la aplicación propuesta se obtuvo el resultado esperado.

Capítulo 4

4. Conclusiones y Observaciones

- El uso de dispositivos lógicos programables para el diseño de *Embedded Systems* se presenta como una alternativa que ofrece varias ventajas por su facilidad de reconfiguración y además por su bajo costo de desarrollo.
- El lenguaje de descripción de Hardware VHDL proporciona una sintaxis amplia y flexible que permite el modelado estructural, en flujo de datos y de comportamiento de hardware. Debido a su estandarización, un código en VHDL puede ser usado por diferentes herramientas y también, puede ser reutilizado en diferentes diseños.
- El software EDK es una herramienta que permite construir un sistema de procesamiento embebido para ser configurado por el usuario. El desarrollo de sistemas de procesamiento embebidos a través del EDK tiene como ventaja la reconfiguración del hardware y software, la flexibilidad y la facilidad de direccionamiento de un procesador IP. Por medio de éste proyecto se logró configurar y direccionar el MicroBlaze utilizando el software EDK (Embedded Development Kit) de Xilinx, con el fin de integrar los módulos en VHDL.
- El MicroBlaze es un procesador IP *softcore* que puede ser implementado totalmente usando síntesis lógica en un FPGA, esto hace que a diferencia de otros procesadores sea flexible, ya que sus parámetros pueden ser modificados en cualquier momento reprogramando el FPGA.
- El proyecto realizado contribuye en la apropiación tecnológica del desarrollo de *Embedded Systems* y adicionalmente pretende hacer un aporte de material pedagógico que pueda servir como apoyo para procesos de enseñanza y aprendizaje en el diseño de Embedded Systems.

5. Referencias Bibliográficas

[1] http://www.xilinx.com/publications/prod_mktg/MicroBlaze_Sell_Sheet.pdf. Revisado en Diciembre 2008.

[2] E. Aguayo. “Procesador Embebido MicroBlaze para FPGAs”. http://www.euroform-ti.org/xilinx/MicroBlaze.pdf. Revisado en Diciembre 2008.

[3] http://www.xilinx.com. Revisado en Noviembre 2008.

[4] http://www.digilentinc.com/

[5] Xilinx XTP013 EDK Concepts, Tools and Techniques, EDK 10.1, Septiembre 18, 2008.

[6] http://www.cannic.uab.es/docencia/DCisII/Apunts/IPcoresIntro.pdf. Revisado en Diciembre 2008.

[7] Xilinx UG454 Spartan-3A DSP Starter Platform User Guide, v1.0, Octubre 3, 2007.

[8] Xilinx DS610 Spartan-3A DSP FPGA Family Data Sheet, Junio 2, 2008.

[9] Xilinx Embedded System Tools Reference Manual, EDK 10.1, Service Pack 3.

[10] S. A. Edwards. “Design and Verification Languages”. IEEE Journal. Noviembre 2004.

[11] National Semiconductor. ADCS7476/ADCS7477/ADCS7478 1MSPS, 12-/10-/8-Bit A/D Converters in SOT-23 & LLP, Septiembre, 2007.

[12] National Semiconductor. DAC121S101 12-bit Micro Power, RRO Digital-to-analog converter, Diciembre 11, 2008.

[13] Fairchild semiconductor. MMBT3904 NPN General Purpose Amplifier, 2001.

ANEXOS

ANEXO A

Este anexo contiene la descripción en VHDL del módulo que controla al periférico PmodAD1™ (Conversor Analógico – Digital) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
 (
   -- Puertos agregados y utilizados por el usuario

              datoin1 : in std_logic; -- Señal digital serial que viene del conversor A/D
              datoin2 : in std_logic; -- Señal digital serial que viene del conversor A/D
              adclk : out std_logic; -- Señal de reloj que va al conversor
              CS : out std_logic; -- Señal que habilita la conversión de datos
              selec : in std_logic; -- Señal que permite al usuario comenzar el proceso de conversión
 );
end entity user_logic;

 architecture IMP of user_logic is
  --Agregado por el usuario
  signal dataA  : std_logic_vector(14 downto 0) := "000000000000000"; -- registro para guardar datos de entrada
  signal dataB  : std_logic_vector(14 downto 0) := "000000000000000"; -- registro para guarda datos de entrada
  signal cnt  : std_logic_vector(3 downto 0) := "0000"; -- contador
  signal seleccion : std_logic := '1'; -- señal de habilitación del conversor
  signal contclk: std_logic_vector(3 downto 0) := "0000"; -- contador para divisor del reloj
  signal Clk: std_logic := '0'; -- señal de reloj del conversor

begin
  -- Procedimiento que divide la frecuencia del reloj que viene del procesador para
  -- enviarla al conversor

process (Bus2IP_Clk)
begin
if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
              if contclk = "0100" then --Se puede modificar para cambiar la frecuencia de muestreo
               contclk <= "00000000";
               Clk <= not Clk;
              else
               contclk <= contclk + '1';
              end if;
end if;
end process;

-- Procedimiento que habilita la conversión de datos

process(Clk,selec)
 begin
              if Clk'event and Clk = '1' then
              if selec = '1' then
              seleccion <= '0';
```

```
            end if;
            end if;
end process;
-- Procedimiento que recibe los bits de entrada y los guarda en un registro
-- finalmente se obtiene el dato convertido

process(Clk)
 begin
            if Clk'event and Clk = '1' then
               if seleccion = '0' then
                        if cnt = "1111" then
                                    slv_reg0(24 to 31) <= dataA(10 downto 3);
                                    slv_reg1(24 to 31) <= dataB(10 downto 3);
                                    cnt <= "0000";
                                    CS <= '1';
                        else
                                    dataA <= dataA(13 downto 0)&datoin1;
                                    dataB <= dataB(13 downto 0)&datoin2;
                                    cnt <= cnt + '1';
                                    CS <= '0';
                        end if;
               end if;
            end if;
end process;

   adclk <= Clk; -- señal de reloj que va al conversor

end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento

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

// Librería del periférico creado

#include "conversor_ad1.h"

//============================================================================
int main (void)
{
            Xuint32 Dato; // se define la variable Dato
            XGpio leds; // se define el puntero LEDs para el periférico Gpio

            XGpio_Initialize(&leds,XPAR_LEDS_8BIT_DEVICE_ID); //Inicializa la estancia XGpio llamando el ID del
            dispositivo

            XGpio_SetDataDirection(&leds,1,0x00000000);// Configura la dirección Entrada/Salida de la
                                                                  //  señal para el canal seleccionado del periférico

while (1)
{
    // Función que me permite leer el dato del periférico AD conectado al procesador
```

```
        Dato = CONVERSOR_AD1_mReadReg(XPAR_CONVERSOR_AD_0_BASEADDR, 0);

        // Procedimiento que compara el valor que viene del periférico con un rango ya establecido

        if (Dato < 0x0000004D)  // Valor en hexadecimal correspondiente a 1 voltio en digital

        XGpio_DiscreteWrite(&leds, 1, 0xFF); // Función que me permite encender los LEDs poniendo
                                             //  un valor de 1 en  todas las salidas
        else
        {
        if (Dato > 0x000000C6) // Valor en hexadecimal correspondiente a 2.6  voltios en digital

        XGpio_DiscreteWrite(&leds, 1, 0xFF); // Función que me permite encender los LEDs poniendo
                                               // un valor de 1 en todas las salidas
        else

        XGpio_DiscreteWrite(&leds, 1, 0x00); // Función que me permite apagar los LEDs poniendo un
                                             // valor de 0 en  todas las salidas
        }
}
}
```

ANEXO B

Este anexo contiene la descripción en VHDL del módulo que controla al periférico PmodDA2™ (Conversor digital-analógico) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
  (
    -- Puertos agregados y utilizados por el usuario

                Dato_salida1 : out  STD_LOGIC; -- Señal serial digital de salida al conversor D/A
                Dato_salida2 : out  STD_LOGIC; -- Señal serial digital de salida al conversor D/A
                sclk : out  STD_LOGIC; -- Señal de reloj que va a los conversores D/A
                sync : out  STD_LOGIC; -- Señal de sincronismo que habilita la conversión de datos

  );
end entity user_logic;

architecture IMP of user_logic is

  -- Agregado por el usuario

            signal registro1 : std_logic_vector(15 downto 0); -- Registro que recibe los datos del procesador
            signal registro2 : std_logic_vector(15 downto 0); -- Registro que recibe los datos del procesador
            signal cont : std_logic_vector(4 downto 0) := "00000"; -- Contador
            signal dato : std_logic := '1'; -- Señal que se habilita cuando está completo el dato
signal contclk: std_logic_vector(4 downto 0) := "00000"; -- Contador para divisor de reloj
            signal clkint: std_logic := '0'; -- Señal de reloj del conversor

begin

-- Procedimiento que divide la frecuencia de la señal de reloj que viene del procesador
-- para enviarla al conversor con una frecuencia menor

process (Bus2IP_Clk)
begin

if (Bus2IP_Clk'event and Bus2IP_Clk = '1') then
            if contclk = "11001" then
                        contclk <= "00000";
                        clkint <= not clkint;
            else
                        contclk <= contclk + '1';
            end if;
end if;
end process;

-- Procedimiento que guarda en registros y envía los datos de salida a los conversores
-- de forma serial, bit a bit
```

```
process(clkint,dato)
begin

        if (clkint'event and clkint = '1') then

                if dato = '1' then
                                registro1 <= slv_reg0(16 to 31);
                                registro2 <= slv_reg1(16 to 31);
                                Dato_salida1 <= '0';
                                Dato_salida2 <= '0';
                        else
                                Dato_salida1 <= registro1(15);
                                Dato_salida2 <= registro2(15);
                                registro1(15 downto 1) <= registro1(14 downto 0);
                                registro2(15 downto 1) <= registro2(14 downto 0);
                        end if;
                end if;
end process;

-- Procedimiento que controla el número de bits enviados y habilita la recepción del dato
-- a convertir que viene del procesador

process(clkint)
begin

                if (clkint'event and clkint = '1') then
                        case cont is
                        when "10000" =>
                                cont <= cont + '1';
                                dato <= '1';
                                sync <= '0';
                        when "10001" =>
                               cont <= "00000";
                                dato <= '1';
                                sync <= '1';
                        when "00000" =>
                                cont <= cont + '1';
                                dato <= '0';
                                sync <= '1';
                        when others =>
                                cont <= cont + '1';
                                dato <= '0';
                                sync <= '0';
                        end case;
                end if;
end process;

sclk <= clkint; -- Señal de reloj que va a los conversores

end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento

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

// Librería del periférico creado
#include "conversor_da1.h"
//============================================================================
int main (void)
{
        int i; // se define la variable i como entera
        int selec; // se define la variable selec como entera
        Xuint32 Dato; // se define la variable Dato
        XGpio leds;  // se define el puntero leds para el periferico Gpio
       XGpio pushb; // se define el puntero pushb para los push buttons del sistema de desarrollo

XGpio_Initialize(&leds,XPAR_LEDS_8BIT_DEVICE_ID); // Inicializa la estancia XGpio
                                                  // llamando el ID del dispositivo
XGpio_SetDataDirection(&leds, 1, 0x00000000); // Configura la dirección Entrada/Salida de la
                                              // señal para el canal seleccionado del periférico

XGpio_Initialize(&pushb, XPAR_PUSH_BUTTONS_DEVICE_ID); // Inicializa la estancia XGpio
                                                       // llamando el ID del dispositivo

XGpio_SetDataDirection(&pushb, 1, 0xFFFFFFFF);  // Configura la dirección Entrada/Salida de la
                                                // señal para el canal seleccionado del periférico

while (1)
{
        // Permite guardar el valor que introducen los push buttons en la variable selec

        selec = XGpio_DiscreteRead(&pushb, 1);

// si se activa el push button entonces se envía una secuencia de valores en un tiempo determinado

        if (selec == 0x1)
        {       for (i=1; i<11111111; i++)
                        {

        // Función que me permite escribir el dato al periférico D/A conectado al procesador
CONVERSOR_DA1_mWriteReg(XPAR_CONVERSOR_DA_0_BASEADDR, 0, 0x0000004D);

         // Función que escribe el valor en los leds
        XGpio_DiscreteWrite(&leds, 1, 0x4D);

                        }
                for (i=1; i<11111111; i++)
                        {
        // Función que me permite escribir el dato al periférico D/A conectado al procesador

CONVERSOR_DA1_mWriteReg(XPAR_CONVERSOR_DA_0_BASEADDR, 0, 0x000000C6);

         // Función que escribe el valor en los leds
          XGpio_DiscreteWrite(&leds, 1, 0xC6);
```

```
                        }
                for (i=1; i<11111111; i++)
                        {
        // Función que me permite escribir el dato al periférico D/A conectado al procesador
CONVERSOR_DA1_mWriteReg(XPAR_CONVERSOR_DA_0_BASEADDR, 0, 0x00000FFF);

        // Función que escribe el valor en los leds
         XGpio_DiscreteWrite(&leds, 1, 0xFF);
                        }
                }
}
}
```

ANEXO C

Este anexo contiene la descripción en VHDL del módulo que controla al periférico PmodOC1™ (Salida colector abierto) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
 (
  -- Puertos agregados y utilizados por el usuario

                    P1              : out std_logic; -- Señal que activa o desactiva el transistor
                    P2              : out std_logic; -- Señal que activa o desactiva el transistor
                    P3              : out std_logic; -- Señal que activa o desactiva el transistor
                    P4              : out std_logic; -- Señal que activa o desactiva el transistor
 );
end entity user_logic;

architecture IMP of user_logic is

 signal data : std_logic_vector (3 downto 0); -- Registro

begin

-- Proceso que guarda el valor del registro interno en data

        process(Bus2IP_Clk)
        begin
                if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                        data <= slv_reg0 (28 to 31);
                end if;
        end process;

        P1 <= data(0);
        P2 <= data(1);
        P3 <= data(2);
        P4 <= data(3);

end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"

// Librería del periférico creado
#include "open_colector.h"

//====================================================================================

int main (void)
{		int i;	// Se define la variable i como entera

		while(1)
		{
		// Se utiliza una sentencia FOR como retardo para mantener el valor y poderlo ver en los LEDs
			for (i=1; i<1111111; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x8);
			}
			for (i=1; i<1111111; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x4);

			}
			for (i=1; i<1111111; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x2);

			}
			for (i=1; i<1111111; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x1);
			}
			for (i=1; i<110000; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x1);

			}
			for (i=1; i<110000; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x2);

			}
			for (i=1; i<110000; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x4);

			}
			for (i=1; i<110000; i++)
			{OPEN_COLECTOR_mWriteReg(XPAR_OPEN_COLECTOR_0_BASEADDR,0,0x8);

			}
}
}
```

ANEXO D

Este anexo contiene la descripción en VHDL del módulo que controla al periférico PmodHB3™ (Puente H) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
 (
  -- Puertos agregados y utilizados por el usuario
          pwm  : out std_logic; -- Señal PWM que controla la velocidad
          D : out std_logic; -- Señal de dirección de giro
 );
end entity user_logic;

architecture IMP of user_logic is

        -- Periodo de la señal PWM: 20 ciclos de reloj
        constant PWM_periodo : std_logic_vector(4 downto 0) := conv_std_logic_vector(20, 5);
        signal cnt: std_logic_vector(4 downto 0) := "00000"; -- Contador
        signal s, reset: std_logic; -- Señales de control

begin

-- Proceso que activa un contador mientras la señal inicio no esté activa, de lo contrario
-- reinicia el contador

        process(Bus2IP_Clk)
        begin
                if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                        if reset = '1' then
                                cnt <= (others => '0');
                        else
                                cnt <= cnt + '1';
                        end if;
                end if;
        end process;
-- Señal de control que se activa cuando el contador llega al valor del periodo de la señal PWM
        reset <= '1' when cnt = PWM_periodo else '0';

-- Señal de control que se activa cuando el contador llega al valor del ancho del pulso
        s <= '1' when cnt = "0"&slv_reg1(28 to 31) else '0';

        -- PWM
                process(Bus2IP_Clk)
                begin
                        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                                        if reset = '1' then
                                                pwm <= '1';
                                        elsif s = '1' then
```

```
                                                pwm <= '0';
                                            end if;
                                    end if;
                            end process;

        D <= slv_reg0(31); -- Señal de dirección de giro

end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xgpio.h"

// Librería del periférico creado
#include "h_bridge.h"

//============================================================================

int main (void)
{       int i; // Se define la variable i como entera
        XGpio led8bits;

        XGpio_Initialize(&led8bits, XPAR_LEDS_8BIT_DEVICE_ID);
        XGpio_SetDataDirection(&led8bits, 1, 0x00000000);

                for (i=1; i<1111111; i++)
                {H_BRIDGE_mWriteSlaveReg0(XPAR_H_BRIDGE_0_BASEADDR,0,0x08);
                XGpio_DiscreteWrite(&led8bits, 1, 0x08);
                } // Se utiliza una sentencia FOR como retardo para mantener el valor que define el
                 // sentido de giro del motor y la velocidad
                for (i=1; i<1111111; i++)
                {H_BRIDGE_mWriteSlaveReg0(XPAR_H_BRIDGE_0_BASEADDR,0,0x00);
                XGpio_DiscreteWrite(&led8bits, 1, 0x00);
                }
                for (i=1; i<1111111; i++)
                {H_BRIDGE_mWriteSlaveReg0(XPAR_H_BRIDGE_0_BASEADDR,0,0x18);
                XGpio_DiscreteWrite(&led8bits, 1, 0x88);
                }
                for (i=1; i<1111111; i++)
                {H_BRIDGE_mWriteSlaveReg0(XPAR_H_BRIDGE_0_BASEADDR,0,0x00);
                XGpio_DiscreteWrite(&led8bits, 1, 0x00);
                }
                for (i=1; i<1111111; i++)
                {H_BRIDGE_mWriteSlaveReg0(XPAR_H_BRIDGE_0_BASEADDR,0,0x08);
                XGpio_DiscreteWrite(&led8bits, 1, 0x08);
                }
}
```

ANEXO E

Éste anexo contiene la descripción en VHDL del módulo que controla al periférico PmodCON3™ (Control de servo motor) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
 (
  -- Puertos agregados y utilizados por el usuario

                        P1      : out std_logic; -- Señal que controla servo motor 1
                        P2      : out std_logic; -- Señal que controla servo motor 2
                        P3      : out std_logic; -- Señal que controla servo motor 3
                        P4      : out std_logic; -- Señal que controla servo motor 4
 );
end entity user_logic;

architecture IMP of user_logic is

-- Periodo de la señal PWM: 2000 ciclos de reloj
        constant PWM_periodo : std_logic_vector(10 downto 0) := conv_std_logic_vector(1999, 11);

        signal cnt: std_logic_vector(10 downto 0) := "00000000000"; -- Contador
        signal servo1, servo2, servo3, servo4, inicio, pe1, pe2, pe3, pe4 : std_logic; -- señales de control
begin
                P1 <= servo1; -- Señal que controla servo motor 1
                P2 <= servo2; -- Señal que controla servo motor 2
                P3 <= servo3; -- Señal que controla servo motor 3
                P4 <= servo4; -- Señal que controla servo motor 4
-- Proceso que activa un contador mientras la señal inicio no esté activa, de lo contrario
-- reinicia el contador

                process(Bus2IP_Clk)
                begin
                if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                        if inicio = '1' then
                                cnt <= (others => '0');
                        else
                                cnt <= cnt + '1';
                        end if;
                end if;
                end process;

-- Señal de control que se activa cuando el contador llega al valor del periodo de la señal PWM

                inicio <= '1' when cnt = PWM_periodo else '0';

-- Señal de control que se activa cuando el contador llega al valor del ancho del pulso
```

```
                pe1 <= '1' when cnt = "000"&slv_reg0(24 to 31) else '0';

                pe2 <= '1' when cnt = "000"&slv_reg1(24 to 31) else '0';

                pe3 <= '1' when cnt = "000"&slv_reg2(24 to 31) else '0';

                pe4 <= '1' when cnt = "000"&slv_reg3(24 to 31) else '0';

-- Proceso que permite generar la señal PWM para cada uno de los servo motores

                process (Bus2IP_Clk)
                begin
                        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                                        if inicio = '1' then
                                                servo1 <= '1';
                                        elsif pe1 = '1' then
                                                servo1 <= '0';
                                        end if;
                         end if;
                end process;

                process (Bus2IP_Clk)
                begin
                        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                                        if inicio = '1' then
                                                servo2 <= '1';
                                        elsif pe2 = '1' then
                                                servo2 <= '0';
                                        end if;
                         end if;
                end process;

               process (Bus2IP_Clk)
                begin
                        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                                        if inicio = '1' then
                                                servo3 <= '1';
                                        elsif pe3 = '1' then
                                                servo3 <= '0';
                                        end if;
                         end if;
                end process;

                process (Bus2IP_Clk)
                begin
                        if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                                        if inicio = '1' then
                                                servo4 <= '1';
                                        elsif pe4 = '1' then
                                                servo4 <= '0';
                                        end if;
                         end if;
                end process;

end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xgpio.h"

// Librería del periférico creado
#include "servo_motores.h"

//============================================================================
int main (void)
{		int i; // Se define la variable i como entera
		Xuint32 valor; // Se define la variable valor
		XGpio push; // Se define el puntero push para los push buttons del sistema de desarrollo

		XGpio_Initialize(&push, XPAR_PUSH_BUTTONS_DEVICE_ID); // Inicializa la estancia XGpio llamando el
                                                                // ID del dispositivo
		XGpio_SetDataDirection(&push, 1, 0xFFFFFFFF);  // Configura la dirección Entrada/Salida de la señal para
                                                        // el canal seleccionado del periférico

while(1)
{		 // Permite guardar el valor que introducen los push buttons en la variable valor
		valor = XGpio_DiscreteRead(&push, 1);

		// Se controla el servo motor con los valores en los PUSH BOTTONS
		  if (valor == 0x0)
			{ // Función que me permite escribir el ancho del pulso al periférico servo_motores

SERVO_MOTORES_mWriteSlaveReg0(XPAR_SERVO_MOTORES_0_BASEADDR, 0, 0xC8);
			}

		  if (valor == 0x1)
			{ // Función que me permite escribir el ancho del pulso al periférico servo_motores

SERVO_MOTORES_mWriteSlaveReg0(XPAR_SERVO_MOTORES_0_BASEADDR, 0, 0x64);
			}

		  if (valor == 0x2)
			{ // Función que me permite escribir el ancho del pulso al periférico servo_motores

SERVO_MOTORES_mWriteSlaveReg0(XPAR_SERVO_MOTORES_0_BASEADDR, 0, 0x96);
			}

		  if (valor == 0x4)
			{ // Función que me permite escribir el ancho del pulso al periférico servo_motores

SERVO_MOTORES_mWriteSlaveReg0(XPAR_SERVO_MOTORES_0_BASEADDR, 0, 0x78);
			}

		  if (valor == 0x8)
			{ // Función que me permite escribir el ancho del pulso al periférico servo_motores

SERVO_MOTORES_mWriteSlaveReg0(XPAR_SERVO_MOTORES_0_BASEADDR, 0, 0xFA);
			}
}
}
```

ANEXO F

Este anexo contiene la descripción en VHDL del módulo que controla al periférico PmodRS232™ (Puerto serial RS232) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
 (
                        reset: in std_logic;
                        tx_start: in std_logic;
                        rx: in std_logic;
                        tx: out std_logic;
 );
end entity user_logic;

architecture IMP of user_logic is
        signal tick: std_logic;
        signal r_reg: unsigned(8 downto 0);
        signal r_next: unsigned(8 downto 0);

        -- Recepcion  ----------------------------------------------

        type state_type is (idle, start, data, stop);
        signal state_reg, state_next : state_type;
        signal s_reg, s_next: unsigned(3 downto 0);
        signal n_reg, n_next: unsigned(2 downto 0);
        signal b_reg, b_next: std_logic_vector(7 downto 0);

        -- Transmisión ----------------------------------------------

        type state_type_tx is (idle_tx, start_tx, data_tx, stop_tx);
        signal state_reg_tx, state_next_tx : state_type_tx;
        signal s_reg_tx, s_next_tx: unsigned(3 downto 0);
        signal n_reg_tx, n_next_tx: unsigned(2 downto 0);
        signal b_reg_tx, b_next_tx: std_logic_vector(7 downto 0);
        signal tx_reg, tx_next: std_logic;
begin

                process (Bus2IP_Clk, reset)
                begin
                        if (reset = '1') then
                                r_reg <= (others => '0');
                        elsif (Bus2IP_Clk'event and Bus2IP_Clk='1') then
                                r_reg <= r_next;
                        end if;
                end process;

        --------- Genera señal síncrona con frecuencia de 16 veces mayor que 9600 baud ----------
```

```
r_next <= (others => '0') when r_reg = 406 else     -- El valor 406 es para obtener una frecuencia de
                                                    --16*9600
r_reg + 1;

tick <= '1' when r_reg = 406 else '0';


-- Recepcion  ------------------------------------------------------------------------------------------------------

process(Bus2IP_Clk, reset)
begin
        if (reset = '1') then
                state_reg <= idle;
                s_reg <= (others => '0');
                n_reg <= (others => '0');
                b_reg <= (others => '0');
        elsif (Bus2IP_Clk'event and Bus2IP_Clk='1') then
                state_reg <= state_next;
                s_reg <= s_next;
                n_reg <= n_next;
                b_reg <= b_next;
        end if;
end process;

process(state_reg,s_reg,n_reg,b_reg,tick,rx)
begin
        state_next <= state_reg;
                s_next <= s_reg;
                n_next <= n_reg;
                b_next <= b_reg;


        case state_reg is
                when idle =>
                        if rx = '0' then
                                state_next <= start;
                                s_next <= (others => '0');
                        end if;
                when start =>
                        if (tick = '1') then
                                if s_reg = 7 then
                                        state_next <= data;
                                        s_next <= (others => '0');
                                        n_next <= (others => '0');
                                else
                                        s_next <= s_reg + 1;
                                end if;
                        end if;
                when data =>
                        if (tick = '1') then
                                if s_reg = 15 then
                                        s_next <= (others => '0');
                                        b_next <= b_reg(6 downto 0) & rx;
                                                if n_reg = 7 then
                                                state_next <= stop;
                                                else
                                                n_next <= n_reg + 1;
                                                end if;
                                else
```

```
                                                                        s_next <= s_reg + 1;
                                                                end if;
                                                        end if;
                                                when stop =>
                                                        if (tick = '1') then
                                                                if s_reg = 15 then
                                                                        state_next <= idle;
                                                                else
                                                                        s_next <= s_reg + 1;
                                                                end if;
                                                        end if;
                                        end case;
                                end process;

                                 slv_reg1(24 to 31) <= b_reg;


-------transmision ----------------------------------------------------------------------------------------------------------------------------

                        process(Bus2IP_Clk, reset)
                        begin
                                        if (reset = '1') then
                                                state_reg_tx <= idle_tx;
                                                s_reg_tx <= (others => '0');
                                                n_reg_tx <= (others => '0');
                                                b_reg_tx <= (others => '0');
                                                tx_reg <= '1';
                                        elsif (Bus2IP_Clk'event and Bus2IP_Clk='1') then
                                                state_reg_tx <= state_next_tx;
                                                s_reg_tx <= s_next_tx;
                                                n_reg_tx <= n_next_tx;
                                                b_reg_tx <= b_next_tx;
                                                tx_reg <= tx_next;
                                        end if;
                        end process;

                        process(state_reg_tx,s_reg_tx,n_reg_tx,b_reg_tx,tick,tx_reg,tx_start,slv_reg0)
                        begin
                                        state_next_tx <= state_reg_tx;
                                                s_next_tx <= s_reg_tx;
                                                n_next_tx <= n_reg_tx;
                                                b_next_tx <= b_reg_tx;
                                                tx_next <= tx_reg;
                                        case state_reg_tx is
                                                when idle_tx =>
                                                        tx_next <= '1';
                                                        if tx_start = '1' then
                                                                state_next_tx <= start_tx;
                                                                s_next_tx <= (others => '0');
                                                                b_next_tx <= slv_reg0(24 to 31);
                                                        end if;
                                                when start_tx =>
                                                        tx_next <= '0';
                                                        if (tick = '1') then
                                                                if s_reg_tx = 15 then
                                                                        state_next_tx <= data_tx;
                                                                        s_next_tx <= (others => '0');
                                                                        n_next_tx <= (others => '0');
                                                                else
```

```
                                                s_next_tx <= s_reg_tx + 1;
                                        end if;
                                end if;
                        when data_tx =>
                                tx_next <= b_reg_tx(7);
                                if (tick = '1') then
                                        if s_reg_tx = 15 then
                                                s_next_tx <= (others => '0');
                                                b_next_tx <= b_reg_tx(6 downto 0) & "0";
                                                        if n_reg_tx = 7 then
                                                        state_next_tx <= stop_tx;
                                                        else
                                                        n_next_tx <= n_reg_tx + 1;
                                                        end if;
                                        else
                                                        s_next_tx <= s_reg_tx + 1;
                                        end if;
                                end if;
                        when stop_tx =>
                                if (tick = '1') then
                                        if s_reg_tx = 15 then
                                                state_next_tx <= idle_tx;
                                        else
                                                s_next_tx <= s_reg_tx + 1;
                                        end if;
                                end if;
                                        end case;
                                end process;
                        tx <= tx_reg;
end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xgpio.h"

// Librería del periférico creado
#include "rs232.h"

//============================================================================
int main (void)
{
XGpio led8bits;
Xuint32 dato;
                XGpio_Initialize(&led8bits, XPAR_LEDS_8BIT_DEVICE_ID);
                XGpio_SetDataDirection(&led8bits, 1, 0x00000000);

while(1)
{
                dato = RS232_mReadSlaveReg1(XPAR_RS232_0_BASEADDR, 0);
                XGpio_DiscreteWrite(&led8bits, 1, dato);

}
}
```

ANEXO G

Este anexo contiene la descripción en VHDL del módulo que controla al periférico PmodAMP1™ (Amplificador de audio) y el código en lenguaje C de la aplicación que se realizó para la validación del módulo descrito.

DESCRIPCIÓN EN VHDL

```
entity user_logic is
port
  (
    -- Puertos agregados y utilizados por el usuario
              inleft: out std_logic;  -- Salida izquierda al amplificador de audio
             inright: out std_logic  -- Salida derecha al amplificador de audio
);
end entity user_logic;

architecture IMP of user_logic is
signal audio: std_logic := '0'; -- Señal interna para generar la señal de audio (Tono)
signal cntclk : std_logic_vector(18 downto 0) := "0000000000000000000"; -- Señal que controla la frecuencia de la
                                                                          señal de audio de salida
begin
process(Bus2IP_Clk) --Proceso que genera la señal de audio (señal cuadrada con frecuencia especificada por el
                              usuario
begin
            if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
                        if cntclk = slv_reg0(13 to 31) then -- Registro interno que define la frecuencia de la señal cuadrada
                                                    cntclk <=  "0000000000000000000";
                                                    audio <= not audio;
                                                   else
                                                    cntclk <= cntclk + '1';
                                                   end if;
                                          end if;
                        end process;
inleft <= audio;
inright <= audio;

end IMP;
```

CÓDIGO EN LENGUAJE C

```
// Librerías básicas de funcionamiento
#include "xparameters.h"
#include "stdio.h"
#include "xutil.h"
#include "xgpio.h"

// Librería del periférico creado
```

```
#include "amplificador.h"

//========================================================================

int main (void)
{           int i; // Se define la variable i como entera
            XGpio led8bits;

XGpio_Initialize(&led8bits, XPAR_LEDS_8BIT_DEVICE_ID);
XGpio_SetDataDirection(&led8bits, 1, 0x00000000);

while(1)
{
for (i=1; i<1111111; i++)
{           // Do (260 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0x1D4C0);
            XGpio_DiscreteWrite(&led8bits, 1, 0x01);
            } // Se utiliza una sentencia FOR como retardo para mantener el tono un tiempo
            for (i=1; i<1111111; i++)
            {// Re (293 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0x19FC5);
             XGpio_DiscreteWrite(&led8bits, 1, 0x02);
            }
           for (i=1; i<1111111; i++)
            {// Mi (330 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0x171E9);
              XGpio_DiscreteWrite(&led8bits, 1, 0x04);
             }
             for (i=1; i<1111111; i++)
             { // Fa (350 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0x15CC5);
              XGpio_DiscreteWrite(&led8bits, 1, 0x08);
              }
             for (i=1; i<1111111; i++)
             {// Sol (392 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0x13767);
              XGpio_DiscreteWrite(&led8bits, 1, 0x10);
              }
              for (i=1; i<1111111; i++)
              {// La (440 Hz)
              AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0x1156E);
               XGpio_DiscreteWrite(&led8bits, 1, 0x20);
              }
              for (i=1; i<1111111; i++)
              {// Si (494 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0xF71B);
               XGpio_DiscreteWrite(&led8bits, 1, 0x40);
              }
              for (i=1; i<1111111; i++)
            {// Do (520 Hz)
            AMPLIFICADOR_mWriteReg(XPAR_AMPLIFICADOR_0_BASEADDR, 0, 0xEA60);
            XGpio_DiscreteWrite(&led8bits, 1, 0x80);
              }
}
}
```