Diseño de los módulos en VHDL que permitan adicionar los periféricos PMB de Digilent al microprocesador *embedded* PicoBlaze, usando la plataforma Starter Kit CoolRunner-II de Xilinx

Ana Maria Olarte Diaz
Edgar German Betancourt Parra

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga, 2009

Diseño de los módulos en VHDL que permitan adicionar los periféricos PMB de Digilent al microprocesador *embedded* PicoBlaze, usando la plataforma Starter Kit CoolRunner-II de Xilinx

Ana Maria Olarte Diaz
Edgar German Betancourt Parra

Trabajo de grado para optar por el título de Ingeniero Electrónico

Director:
Mie (c) Carlos Augusto Fajardo Ariza

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICOMECÁNICAS
ESCUELA DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga, 2009

Resumen

TÍTULO: Diseño de los módulos en VHDL que permitan adicionar los periféricos PMB de Digilent al microprocesador *embedded* PicoBlaze, usando la plataforma Starter Kit CoolRunner-II de Xilinx[1].

AUTORES: Olarte Diaz, Ana María y Betancourt Parra, Edgar German[2].

PALABRAS CLAVES: *Embedded systems*, CPLD, Starter Kit coolRunner II, PicoBlaze, VHDL, Xilinx, Periféricos (PMB), soft-core.

DESCRIPCIÓN:

Durante el presente proyecto se logró el diseño de los módulos en VHDL que permiten adicionar los periféricos PMB de Digilent al microcontrolador *embedded* PicoBlaze, usando la plataforma Starter Kit CoolRunner-II de Xilinx, demostrando la posibilidad de implementar *embedded systems* sobre CPLDs y presentar una opción en la utilización de software y hardware para los diseños de *embedded systems*.

Este proyecto se inició con la revisión bibliográfica de los siguientes temas:

- *Embedded systems.*
- Familia de CPLDs CoolRunner II de Xilinx.
- Plataforma de desarrollo Starter Kit CoolRunner-II.
- Procesadores sotf-core y específicamente el PicoBlaze de Xilinx
- Comunicación entre periféricos y procesador
- Herramientas CAD necesarias para el desarrollo e implementación de un *embedded system*, específicamente ISE de Xilinx, PBlaze y ASM.exe.

El diseño de cada módulo se inició con un estudio de su funcionamiento, para luego hacer su respectiva descripción y simulación en VHDL. Después de comprobar el correcto funcionamiento del módulo, se propuso una aplicación en la que se empleó tanta programación en software y hardware y de esta manera dejar una base para quienes se inician en el tema de *embedded systems* sobre CPLDs.

Cabe resaltar que en este proyecto se mostró una alternativa de diseño, en la cual se emplearon dos plataformas para obtener una mejor perspectiva de un *embedded system*, presentándose la diferenciación entre software y hardware; brindando una posibilidad para el desarrollo de diseños en los cuales se requiera emplear una mayor capacidad de recursos.

[1]Trabajo de grado.

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

Abstract

TITLE: Modules design in VHDL which let the addition of the peripheral PMB of Diligent to the embedded micro processer Pico Blaze using a platform Starter Kit CoolRunner-II de Xilinx[3].

AUTHORS: Olarte Diaz, Ana María y Betancourt Parra, Edgar German[4].

KEY WORDS: Embedded system, CPLD, Starter Kit Cool Runner II, PicoBlaze, VHDL, Xilinx, Peripheral (PMB), soft-core.

DESCRIPTION:

During this Project it was achieved to design VHDL modules which let adding the periphery PMB of Diligent to the micro controller embedded Pico Blaze using a platform Starter Kit CoolRunner-ll of Xilinx that shows the possibility to implement embedded systems on CPLDs and have the option of using a software and a hardware to design embedded systems.

This project started with the bibliographic checking of the following topics:

- Embedded systems.
- CPLDs Cool Runner II de Xilinx family.
- Development platform Starter Kid Cool Runner-II.
- Soft-core Processer, specifically Pico Blaze of Xilinx.
- Connections between peripheral and processer.
- Necessary CAD tools to build up and execute an embedded system specifically ISE of Xilinx, PBlazeIDE and ASM.exe.

To design each module it started revising its function, in order to make the corresponding description and simulation in VHDL. After verifying the correct functioning of the module it was suggested an application in which was used a software program and hardware as well and in this way let a data base for those who start working on embedded systems on CPLDs.

It's important to bear in mind that in this project was showed an alternative design in which was applied two platforms so as to get a better perspective of an embedded system, showing the difference between software and hardware and providing the possibility to build up designs that have need of using a larger capacity of resources.

[3] Degree Work.

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

Ana María

A Dios por ser mi guía y apoyo para salir adelante.

A mis padres ya que gracias a ellos soy quien soy hoy en día, por su constante apoyo, comprensión, enseñarme a luchar por lo que anhelano y ser mi soporte a pesar de la distancia.

A mi hermana por su aliento y ser mi mejor amiga.

A mis amigos por su paciencia, por los momentos vividos y su amistad incondicional.

Edgar German

A mis padres por su constante apoyo

A mis hermanos por su colaboración y constante ánimo

A mi novia por su voz de aliento y constante comprensión

Agradecimientos

Un especial agradecimiento a nuestros padres por su apoyo incondicional durante toda nuestra vida.

Agradecemos a todas las personas que han contribuido en el proceso de formación como ingenieros.

A la Universidad Industrial de Santander por brindarnos la oportunidad de ingresar al mundo laboral con una formación integral.

Al profesor Carlos Augusto Fajardo por su acompañamiento, guía y colaboración al desarrollar el proyecto.

Al profesor Jorge Hernando Ramón Suaréz por su colaboración y apoyo a lo largo de la carrera.

Al profesor William Alexánder Salamanca Becerra por su motivación y orientación desinteresada en el desarrollo del proyecto.

Índice de contenido

Índice de figuras

Índice de tablas

Introducción

Los *embedded systems* (ES)[5] se han convertido en el principal desarrollo tecnológico de los últimos años, tanto así que han generado una nueva era tecnológica que desde hace varios años se ha denominado la Era Post PC [51]. Esta nueva era se ve influenciada por el afán de satisfacer las necesidades, en un principio, básicas para pasar a resolver problemas más específicos a cada usuario. Los ES se pueden encontrar en teléfonos celulares, cámaras fotográficas, ascensores, ropa, juguetes y reproductores de audio [10].

Los dispositivos lógicos programables (PLD[6]) están jugando un papel importante en el diseño de este tipo aplicaciones [2], lo que ha llevado a muchas facultades de ingeniería en el mundo a hacer una apropiación tecnológica del diseño de ES basado en PLDs como las FPGAs[7] o CPLDs[8] [52] [53].

Motivados con las tendencias desarrolladas a nivel mundial en el tema de ES y basados en las necesidades de los usuarios, la Universidad Industrial de Santander y especialmente la Escuela de Ingeniería Eléctrica, Electrónica y Telecomunicaciones, se manifiesta la necesidad de avanzar en el desarrollo de sus aplicaciones embebidas sobre CPLDs, obteniendo un paso importante en el desarrollo de esta tecnología [51] [10].

Para el diseño de un ES, se requiere de la utilización de periféricos para comunicarse con el medio, estos periféricos son programados en VHDL[9] sobre la

[5] ES: sub-sistema electrónico programable que generalmente hace parte de un sistema más grade y que han sido diseñados para una tarea específica.
[6] Programmable Logic Device.
[7] Field Programmable Gate Array.
[8] Complex Programmable Logic Device.
[9] VHSIC (Very High Speed Integrated Circuit) Hardware Description Language.

plataforma Starter Kit CoolRunner II. A esta plataforma se le pueden adicionar el paquete de módulos periféricos (PMB[10]), para ser controlados por un CPLD.

La base de este proyecto es realizar la comunicación de los PMB y la plataforma, utilizando el procesador soft-core PicoBlaze, haciendo un aporte a la apropiación tecnológica en el diseño de los ES y contribuyendo para futuros interesados en la profundización del estudio de ES, además se pretende que este proyecto sea una base para futuras asignaturas en el dominio de los *embedded systems*.

[10] Peripheral Module Bundle.

Capitulo 1

Dispositivos Lógicos Programables: PLDs Y CPLD

Los dispositivos programables son aquellos circuitos de propósito general que no poseen una estructura interna fija, la estructura puede ser modificada por el usuario para implementar el diseño de sistemas digitales, ver figura 1.

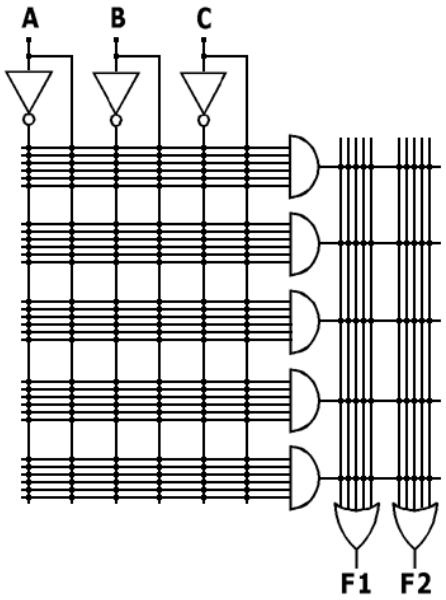

Figura 1. Estructura interna de un dispositivo programable [2].

1.1 Plataforma de desarrollo Starter Kit CoolRunner – II

La plataforma de desarrollo Starter Kit CoolRunner II, es un sistema de desarrollo el cual permite implementar, evaluar y poner en práctica las aplicaciones o diseños deseados. Cada plataforma cuenta con una serie de herramientas, como leds, pulsadores, displey 7 segmentos, puertos de expansión, facilitando la implementación y supervisión de aplicaciones.

La plataforma de desarrollo Starter Kit CoolRunner – II fue implementada con circuitos para el manejo de CPLDs, cuenta con un puerto USB, para la alimentación de la tarjeta y establecer la comunicación entre el usuario y el CPLD, como se puede observar en la figura 2 [1].

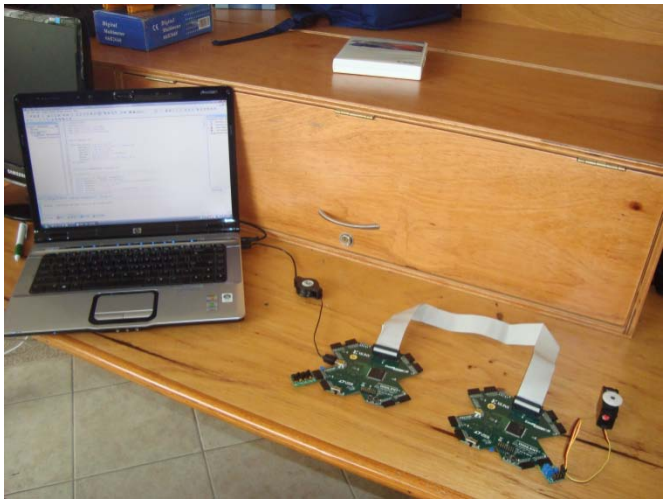

Figura 2. Conexión tarjeta con la computadora.

El CPLD cuenta con 75 señales para ser enviados a los seis conectores de expansión, haciendo que los diseños sean fáciles de expandir y con posibilidad de conexión con periféricos [1].

Las características de la tarjeta son [1]:

- Es robusta, económica y portátil.
- El CPLD cuenta con 256 macroceldas en un paquete de TQ-144.
- Cuando está encendida el puerto USB sirve para la transferencia de datos.
- En la conversión de A/D cuenta con 16 bits, para conversiones en tiempo real durante el proceso de conversión (los datos son enviados para el PC por el cable USB).
- Tiene la facilidad de usar dos oscilador según la necesidad, un de estos es un oscilador de silicio regulable (1000/100/10KHz) y el otro oscilador de cristal de 48 MHz.
- Cuenta con 12 leds y dos pulsadores para I/O.
- El regulador provee de 3.3 V.
- La I/O que no se utilicen deben ser puestas a tierra para minimizar el consumo de energía.

La configuración de inicialización se puede realizar por esquemas de circuito o por archivos HDL[11], que usa el software de WebPad de ISE desde Xilinx y la trasmisión de datos es realizada por el puerto USB [1]. Una vez la tarjeta es encendida, el CPLD queda listo para ser programado o para utilizar la ultima configuración realizada [1].

La tarjeta cuenta con un medidor constante de corriente del CPLD llamado X-Meter, basado en el convertidor de señal LTC248016 de tecnología lineal, el cual utiliza un puerto de SPI[12] para enviar muestras de datos al PC, y el SPI comparte el pin con el puerto de programación JTAG [1]. Para esto se cuenta con un

[11] Lenguaje de Descripción de Hardware.
[12] Serial Peripheral Interface Bus: es un bus de tres líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo, por lo que este tipo de comunicación serial es full duplex. Dos de estas líneas trasfieren los datos (una en cada dirección) y la tercer línea es la del reloj [43].

interruptor para seleccionar entre el X-Meter y el puerto de programación de JTAG [1].

Para medir la corriente y la temperatura de la tarjeta, ella cuenta con un convertidor analógico-digital LTC2480 de tecnología lineal [1]. En donde luego de ser adquiridos los datos, estos son enviados al PC por el puerto USB de la tarjeta [1]. El X-Meter adquiere una medida de temperatura y nueve muestras en tiempo real del consumo de corriente cada dos segundos y transfiere las muestras a una memoria intermedia de datos en el PC y produce una grafica con los datos obtenidos, mostrando los datos más recientes y con los máximos y mínimos [1].

El suministro de energía para la tarjeta es por el puerto USB esencialmente, o de un suministro externo conector al jumper JP3 [1]. El jumper JP2 da la posibilidad de escoger si la alimentación de la plataforma es por USB o por alimentación externa [1].

La energía necesaria para los periféricos conectados a los conectores de expansión, es suministrada desde la alimentación de la tarjeta y pasa a un regulador de voltaje [1]. Este regulador maneja voltajes entre 3.6V a 9V [1].

La tarjeta incluye un oscilador de silicio regulable por el usuario que causa una señal de reloj 1 MHz, 100 KHz o 10 KHz, mediante el pin J11 [1]. También cuenta con un oscilador fijo SMT y un reloj fijo de 48 MHz (1). A continuación se muestra la asignación de pines según el reloj con el que se desee trabajar.

10, 100 or 1000 kHz (J11)	P38
48 MHz	P30
Optional clock at IC4	P32

Figura 3. Asignación para seleccionar el reloj [1].

Los botones incorporados en la tarjeta producen uno "1" al CPLD cuando el botón está hundido [1]. Cada botón se encuentra en serie con una resistencia como

protección contra los cortocircuitos (un cortocircuito puede ocurrir cuando un botón es asignado como salida) [1]. A continuación se muestran los pines relacionados a los dos botones.

Btn0	P61
Btn1	P60

Figura 4. Asignación de los botones [1].

Los doce leds rojos incorporados en la tarjeta son suministrados como indicadores, y son configurables por el usuario. Dos LED de color ámbar indican que la tarjeta esta conecta al cable USB (LD13) y que este está suministrando la energía a la tarjeta [1]. La asignación para cada uno de los leds son:

LED(0)	P131
LED(1)	P105
LED(2)	P88
LED(3)	P87
LED(4)	P86
LED(5)	P74
LED(6)	P82
LED(7)	P44
LED(8)	P26
LED(9)	P20
LED(10)	P6
LED(11)	P138

Figura 5. Asignación de los leds [1].

Los conectores de expansión proveen VDD, GND, y cuatro señales de CPLD únicas para el envió de los datos de control, a los periféricos que sean conectados [1].

1.2 Arquitectura del CPLD XC2C256

El CPLD es un dispositivo que cuenta con 256 macroceldas, diseñadas para solicitudes de poco y gran rendimiento, proporcionando una alta velocidad en la comunicación y bajo consumo de potencia [1]. Las macroceldas del CPLD esta diseñadas para ser energizadas en un rango desde 1.5 V a 3.3 V [1].

Este dispositivo consta de 16 bloques funcionales interconectados por una matriz de interconexión avanzada de baja potencia[13] [1]. Los bloques funcionales constan de 40 a 56 P-tem PLA y 16 macroceldas que contienen numerosas configuraciones para permitir combinaciones o registros de operación [1].

Estos registros pueden ser programados como flip-flop tipo D o T ó como latch tipo D [1]. También se cuenta con múltiples señales de reloj, tanto globales como locales, configurado en una base de macrocelda [1]. Estas frecuencias de reloj son configuradas por el usuario, según la aplicación, por medio de un conjunto de pines externos [1].

Una característica del CPLD es que los flip-flop pueden ser de flancos duales, permitiendo un alto rendimiento en la operación síncrona basándose en cronometrar la frecuencia interior para ayudar a reducir el consumo de energía en el dispositivo [1].

Otra característica con la que cuenta el CPLD es la posibilidad de desactivar las contribuciones del CPLD que no son necesarias durante ciertos procesos de forma selectiva, lográndose el más bajo consumo de energía [1].

[13] AIM Advanced Interconnect Matrix [2].

Los CPLDs de la CoolRunner – II están fabricados con tecnología de 0.18 μm y emplean RealDigital, una técnica de diseño que emplea tecnología CMOS[14] tanto en el diseño como en la fabricación [1]. Esta tecnología emplea una cascada de compuertas CMOS para poner en práctica la suma de productos [1].

Algunas características del CPLD son [1]:

- La demora entre pin y pin es de 5 ns.
- Su estructura es óptima para síntesis de lógica.
- Dispone de múltiples opciones:
    - 100 pines VQFP con 80 usos de I/O.
    - 144 pines TQFP con 106 usos de I/O.
    - 132 – ball CP (0.5 mm) BGA con 118 usos de I/O.
    - 208 pines PGFP con 173 usos de I/O.
    - 256-ball FT (1 mm) BGA con 184 usos I/O.
- Rapidez en la programación del sistema.
- 1.8 V ISP[15] usando interface IEEE 1532 (JTAG).

[14] Complementary Metal Oxide Semiconductor, es el tipo de tecnología de semiconductores que utilizan circuitos NMOS (polaridad negativa) y PMOS (polaridad positiva), emplean menos energía que los chips que usan solo un tipo de transistor [7].
[15] In-System Programmability [3].

1.3 Herramientas de diseño asistido por computador (CAD)

El diseño asistido por computador, conocido por sus siglas CAD[16], es el uso de un amplio rango de herramientas computacionales utilizadas por ingenieros, arquitectos y otros profesionales del diseño en sus respectivas actividades [3].

Mediante el empleo del CAD se logra un avance importante en el diseño, desarrollo y fabricación de proyectos, ya que se minimiza errores de fabricación, precio y tiempo entre que se concibe la idea y se hace realidad [3].

Los CAD conllevan a la eliminación de errores producidos por los elementos e implementaciones discretas y de esta manera se tiene una solución a los requerimientos de funcionamiento, debido a la complejidad que están alcanzando los diseños digitales [4].

Estos diseños implementan multitud de funciones, las cuales deben realizarse de forma rápida y precisa, por lo que resulta necesario el empleo de circuitos integrados digitales programables con mayor complejidad que los ampliamente difundidos CPLD [3].

Actualmente, un nuevo sistema de diseño tiende a implantarse. Una vez hechas las especificaciones, el sistema es dividido en grandes bloques como pueden ser memorias, microprocesador, PLD y FPGA, se realiza una descripción de alto nivel a través de un esquema o lenguaje de descripción lógica para posteriormente simular el diseño, cuando la simulación es correcta se realiza el diseño de la placa que contendrá los diferentes elementos, mientras la placa está siendo fabricada, se depura el diseño, pero en la mayoría de los casos esto no afecta a la placa sino a los circuitos programables incluidos en ella.

[16] Computer Aided Design [3].

1.3.1 Herramientas de diseño Xilinx de ISE

El entorno de diseño **Xilinx ISE** es una herramienta que permite realizar un diseño completo basado en lógica programable (tanto CPLD como FPGA), posee un aspecto similar al de los entornos de programación actuales como puede ser Visual Basic o Visual C, es decir, incluye todas las etapas necesarias como son [5]:

- La entrada de diseño, bien a través de captura esquemática, lenguajes de descripción hardware como ABEL, VHDL o Verilog, o representación gráfica de diagramas de estado.
- Herramientas de verificación para la obtención de una simulación del sistema, tanto a nivel funcional como de estimación de retardos. La herramienta empleada se denomina ModelsimXE. Por otro lado, también se facilita la generación de bancos de prueba para la verificación mediante la herramienta HDL Bencher[17].
- Herramientas de implementación donde se permite la especificación de restricciones o indicaciones para realizar una implementación óptima sobre el dispositivo lógico programable especificado. Esta herramienta incluye tres etapas principales en el diseño: Translate[18], Map[19], Place & Route[20].
- Herramientas de programación, que permiten descargar el diseño sobre el dispositivo físico, ya sea en una placa de evaluación o bien en la placa definitiva mediante la programación in-situ (en sistema) a través de la programación JTAG. De este modo, es posible probar y depurar el sistema

[17] HDL Bencher: es una herramienta de simulación que verifica paso a paso el diseño en condiciones de entrada específico, comprobando el funcionamiento esperado según la entrada [8].

[18] Convertir el netlist de entrada y restricciones en un archivo genérico para Xilinx [5].
[19] Inscribir el diseño en los recursos lógicos disponibles [5].
[20] Programación temporal de los recursos en el dispositivo [5].

sobre el hardware de forma rápida y flexible, permitiendo tantos cambios como sean necesarios.

En la figura 6 se muestra de forma global los procesos que se llevan a cabo durante el diseño de sistemas basados en lógica programable.

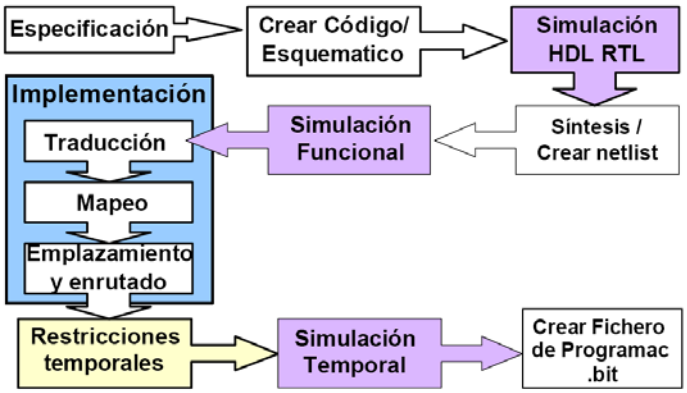

Figura 6. Proceso de diseño basado en lógica programable [5].

Este entorno permite combinar las diferentes técnicas de diseño para facilitar la labor de descripción del diseño. Además, se permite la inclusión de restricciones para optimizar el proceso de implementación y adaptarlo a las necesidades del diseño, como ejemplo, inclusión de restricciones temporales para determinadas señales, restricciones de ubicación de la lógica en una determinada zona de la matriz programable, o bien inclusión de opciones de particularización en elementos hardware como asignación de pines, líneas de reloj específicas, etc. [5].

Por otro lado, el conocimiento de este tipo de entornos permite ser capaces de emplearlo en múltiples diseños, ya que independientemente de la complejidad del diseño, si éste está destinado a un dispositivo programable, ya sea CPLD o FPGA, será posible realizarlo mediante el mismo software, por lo que una vez dominado, el proceso de diseño de nuevos sistemas resulta mucho más rápido [5].

Capítulo 2

Procesadores *Soft-core*

Los procesadores hacen parte de un *embedded system*, es el encargado de desarrollar procesos matemáticos e instrucciones que tenga en la memoria, facilitando la implementación y control de periféricos.

2.1 Módulos IP soft-core y hard-core

Desde su primera aparición hasta la actualidad, los procesadores han ido evolucionando a pasos agigantados, lográndose el paso de procesadores hard-core a soft-core. Los procesadores hard-core son los procesadores tradicionales llevados al silicio y que encontramos de manera física y poseen un área adjunta de lógica programable, que es la única configurable [12][13].

Los procesadores soft-core son aquellos que son definidos mediante una descripción en HDL, son compilados y cableados de distinta manera según la aplicación, obteniendo un mejor rendimiento y reduciendo costos de implementación [14]. Al emplear este tipo de procesadores permite incluir sólo los recursos que sean necesarios o poder agregar recursos adicionales, son los indicados para realizar tareas de baja velocidad pero elevada complejidad y son poco eficientes para alcanzar frecuencias altas como los hard-core.

Los soft-core se encuentran de diferentes tipos y potencia, desde pequeños microcontroladores hasta procesadores de 32 bits [13][14].

2.2 PicoBlaze

Una de las tendencias a nivel mundial que está tomando mucha fuerza es la implementación de microcontroladores Soft-Core sobre CPLDs, estos ofrecen la posibilidad de crear aplicaciones específicas para ser realizadas por los procesadores [11].

Arquitectura

El PicoBlaze es un microcontrolador que acepta 49 instrucciones, 8 registros de ocho bit, 256 puertos direccionables, para ser empotrado en el dispositivo XC2C256-5TQ144 de la tarjeta CoolRunner-II, del cual utiliza 212 macroceldas (83%), ofreciendo un bajo consumo de corriente y un alto desempeño [11]. Una característica importante de este microcontrolador es que cada una de las instrucciones realizadas por el microcontrolador tiene una duración de dos ciclos de reloj [11]. El diagrama de bloques del PicoBlaze se muestra en la figura 7.

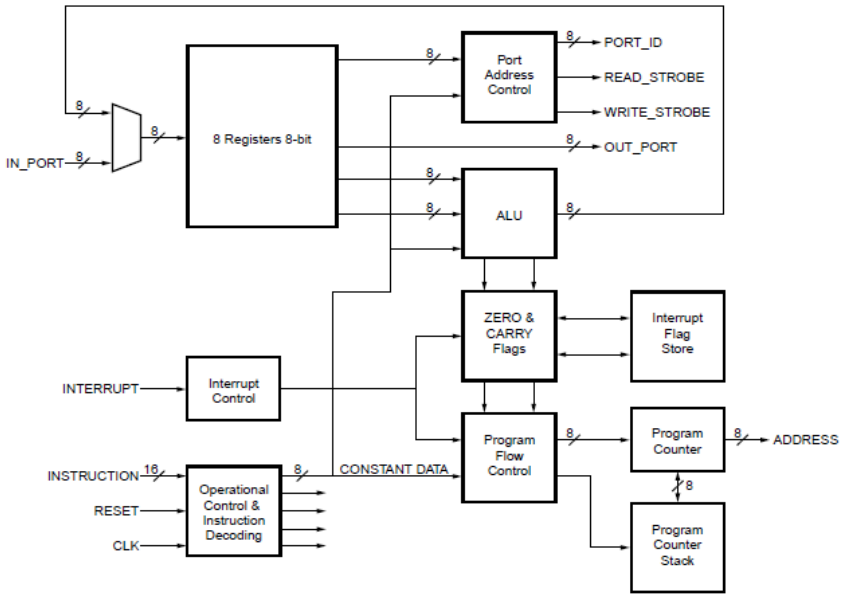

Figura 7. Arquitectura del PicoBlaze [11].

Bloques funcionales del PicoBlaze:

- ❖ Registros de propósito general

El PicoBlaze cuenta con ocho registros de propósito general de 8 bit, s0 a s7. Estos registros no están reservados para tareas específicas y las operaciones de registro son completamente generales [11].

- ❖ Unidad lógica aritmética

La Unidad Lógica Aritmética (ALU) realiza todas las operaciones simples en paquetes de 8 bit, incluyendo [11]:

- Operaciones de aritmética básica (suma y resta), tiene la opción de incluir la bandera CARRY como una entra para dar soporte a operaciones que requieren más de 8 bits.

- Operaciones lógicas (AND, OR y XOR)
- Operaciones de rotación y desplazamiento

Todas las operaciones son realizadas usando un operador indicado por algún registro específico (sX) y el resultado es devuelto al mismo registro. Si una instrucción requiere un segundo operando, entonces el segundo operando es un segundo registro (sY) o una de constante inmediata de 8 bits (kk) [11].

❖ Banderas

Las operaciones de la ALU afectan las banderas de cero (ZERO) y acarreo (CARRY). La bandera ZERO indica que el resultado de la última operación fue cero [11].

La bandera CARRY indica varias condiciones dependiendo de la instrucción ejecutada. Es establecida cuando en el desborde de operaciones aritméticas; también, es usada para capturar el bit movido afuera del registro durante una instrucción desplazamiento o rotación. El estado de las banderas puede ser usado para determinar ejecuciones de secuencias del programa, usando condicionales o no-condicionales en el control de flujo de programa [11].

La bandera de interrupción INTERRUPT_ENABLE habilita la entrada de interrupciones [11].

❖ Entradas / Salidas

El PicoBlaze tiene 256 puertos de entrada / salida. Una operación de lectura es indicada por el puerto IN_PORT y es almacenado en un registro específico y a su vez se envía un pulso en READ_STROBE; de igual manera, cuando está realizando la operación de salida, el contenido es escrito en un registro especifico,

al puerto OUT_PORT, se activa un pulso en WRITE_STROBE. El manejo de estas pulso son una manera de especificar que se está adquiriendo o pasando el dato a otro sistema, respectivamente [11].

- Contador de Programa

El contador de programa (PC) apunta a la siguiente instrucción a ejecutar. El PC se incrementa predeterminadamente a la siguiente ubicación de la instrucción cuando se ejecuta una instrucción. Solo las instrucciones JUMP, CALL, retorno (RETURN) y retorno de interrupción (RETURNI) y los eventos de interrupción y reset modifican el comportamiento predeterminado. El PC no puede ser modificado directamente por el código de la aplicación [11].

- Pila CALL/RETURN

La pila "CALL/RETURN" almacena 4 direcciones de instrucción, lo que permite anidar un "CALL" de secuencias de hasta 4 niveles de profundidad. La pila se utiliza también durante la operación de una interrupción, al menos uno de estos niveles deben reservarse cuando la interrupción se encuentra activa [11].

Cuando la pila está llena sobrescribe el valor más viejo. Cada instrucción de llamada debe especificar la dirección de 8 bit como un valor en hexadecimal, para simplificar esto es válido utilizar etiquetas [11].

Es importante aclarar que no existen instrucciones de control para la pila y no se le reserva memoria de programa [11].

- Interrupción

El PicoBlaze tiene una entrada INTERRUPT y puede usarse en combinación con otras señales si se requiere. Esta entrada permite manejar eventos externos asíncronos [11].

Una interrupción obliga al microcontrolador a llamar la última posición de la memoria de programa, “CALL FF”, donde el usuario define la acción a seguir. Al realizar la interrupción se almacena el estado de las banderas y se deshabilita cualquier otra interrupción. El comando de retorno de interrupción (RETURNI) se asegura que al final de una rutina de interrupción se restaure el estado de las banderas y se habilite el control de interrupción [11].

- Reset

La entrada RESET coloca al PicoBlaze en estado inicial en donde el PC es puesto en la dirección 0, las banderas limpiadas, interrupciones deshabilitadas, y la pila CALL/RETURN es reseteada [11].

- Constantes

La designación de constantes es empleada en un programa para designar valores, direcciones de puertos y valores de dirección. El uso de las constantes no genera espacio adicional en el tamaño del programa o en su ejecución [11].

Señales y conexión del microcontrolador PicoBlaze:

El PicoBlaze cuenta con señales de entrada y salida que le permiten comunicarse con la memoria de programa y con el exterior. En la figura 8, se muestra el diagrama de interconexión.

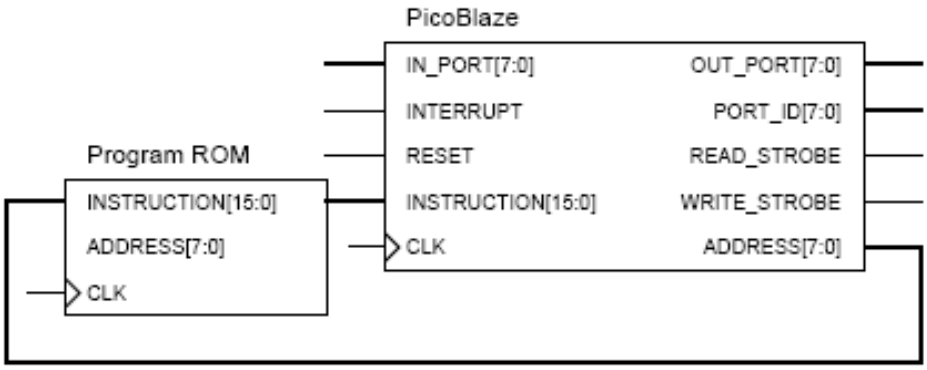

Figura 8. Diagrama de conexión entre la memoria de programa y el PicoBlaze [3].

IN_PORT [7:0]: presenta los datos de entrada validos en este puerto durante una instrucción de entrada. Los datos son capturados en el flanco de reloj de subida [11].

INTERRUPT: si la bandera INTERRUPT_ENABLE es establecida para el programa, genera un evento INTERRUP. Si la bandera INTERRUP_ENABLE es desactivada, esta entrada es ignorada [11].

RESET: está señal es empleada para resetear el microcontrolador y genera la inicialización de todos los procesos realizados [11].

CLK: la frecuencia del reloj está determinada por la configuración que se establezca en la tarjeta. Todos los elementos que responden al reloj, son sincronizados por el flanco de subida de reloj [11].

OUT_PORT [7:0]: durante dos ciclos de reloj los datos de salida aparecen en este puerto [11].

PORT_ID [7:0]: las dirección del puerto de E / S que aparecen en este puerto son mantenidos por dos ciclos de CLK durante una instrucción de salida o entrada [11].

READ_STROBE: al establecerse en alto esta señal indica que los datos de entrada en el puerto IN_PORT [7:0] fueron capturados a un registro de datos durante una instrucción INPUT [11].

WRITE_STROBE: al establecerse en alto esta señal valida los datos de salida en el puerto OUT_PORT [7:0] durante una instrucción de salida [11].

2.3 Herramienta de desarrolla para el PicoBlaze

En el diseño de una aplicación utilizando el Picoblaze se necesita de un código fuente, este es generado en assembler por medio de dos herramientas que son el pBlazeIDE y ASM.EXE.

- pBlazeIDE:

El pBlazeIDE es un entorno que ofrece la posibilidad de compilar y generar un archivo en VHDL correspondiente al programa en Assembler, adicionalmente permite la simulación de las señales de entrada y salida. Este software está equipado con varias versiones de PicoBlaze pero para ser usado sobre FPGA y no sobre CPLD (que es nuestro caso), por lo tanto en este proyecto se utilizó únicamente simulación.

- ASM.EXE:

El ASM.EXE al igual que el pBlazeIDE ofrece la posibilidad de compilar y generar un archivo en VHDL correspondiente al programa en Assembler, tanto para FPGAs como CPDLs. Una desventaja de este entorno es que no permite simulaciones. Al ejecutar este software muestra el numero de errores originados por el empleo de códigos no validos y después de ser corregidos genera cinco archivos [11]. Los cuales son:

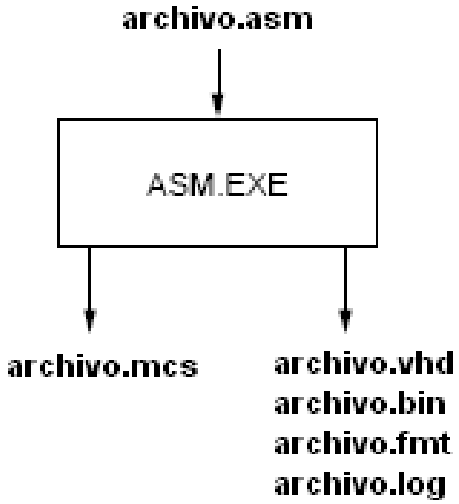

Figura 9. Diagrama de bloques del funcionamiento del ASM.EXE.

- Archivo.vhd: este es un módulo en VHDL para la memoria de programa generado y es apropiado para la síntesis y simulación [11].
- Archivo.bin: este es un archivo en binario para la memoria de programa en formato hexadecimal y es apropiado para la depuración de programa [11].
- Archivo.fmt: en este archivo se encuentra el programa original con un orden distinto. El analizar este archivo es válido para verificar que el programa haya sido interpretado de la manera prevista [11].

- Archivo.log: este archivo le indica al ensamblador que el proceso se efectuó y le muestra los posibles errores generados durante el proceso [11].
- Archivo.mcs: este archivo contiene el código en binario para la memoria de programa en el formato MCS-86 de Intel y también puede emplearse es una memoria externa si se necesita [11].

Capítulo 3

Arquitectura Seleccionada

De manera general un *embedded system* consta de un procesador, software, memoria, bloques periféricos de entrada y salida como se puede observar en la figura 10. La base del proyecto es el manejo de los periféricos de la tarjeta mediante el microprocesador PicoBlaze, el cual es sintetizado sobre el CPLD de la tarjeta Coolrunner-II. Para lograr esto se diseñan los módulos en VHDL para el direccionamiento de los periféricos y se realiza una aplicación.

3.1 Distribución de las plataformas

La arquitectura del proyecto cuenta con dos plataformas[21], una de ellas contiene el procesador con su respectiva memoria y la segunda contiene el módulo de control del periférico. Estas dos plataformas se comunican por medio de un bus de datos síncrono, utilizado una misma fuente de reloj que es enviada de la primera plataforma hacia la segunda, esta arquitectura se presenta a manera de diagrama de bloques en la figura 10 y de manera física en la figura 11.

[21] La plataforma utilizada en el proyecto es la X Board Coolrunner II

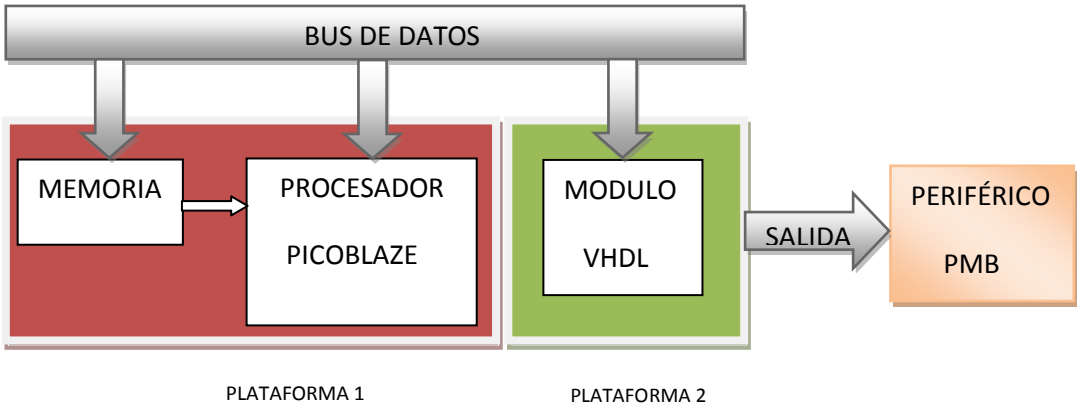

Figura 10. Arquitectura general de las aplicaciones.

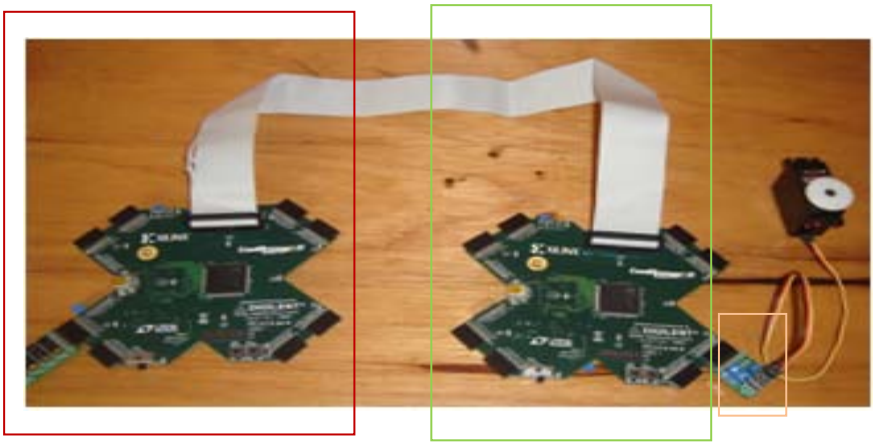

Figura 11. Arquitectura general de las aplicaciones (física).

La comunicación de las tarjetas se realizó empleando el conector J9, de 26 pines que traen incorporado las tarjetas; de los cuales uno es una línea de alimentación y otro es la referencia a tierra, VCC3V3 y GND, respectivamente.

Mediante este conector se creo un bus de datos, por el cual se transmiten la señal de reloj y los demás son empleados según la aplicación.

3.2 Programación de los CPLDs

La programación de las tarjetas se realizó empleando el software Xilinx de ISE, que brinda la posibilidad de generar esquemáticos y diagramas de bloques de los diseños planteados, y de esa manera observar las conexiones entre bloques y analizar posibles errores, como se describió en el capítulo 1.

En las implementaciones es necesario describir la interconexión entre la memoria (el archivo .vhdl arrojado por el ensamblador[22]) y el microprocesador, en un archivo VHDL denominado de alto nivel. Esta interconexión se realiza mediante la unión de los puertos:

- La entrada de la memoria *address*, con la señal de salida del procesador *address*.
- La entrada del reloj *clk* tanto de la memoria como la del microprocesador con el reloj interno de la plataforma.
- La salida de la memoria *instruction* con la entrada del procesador *instruction*.
- En la entrada *in_port* del microprocesador, se conecta la entradas que se necesiten para la aplicación que se este implementando.
- La entrada *reset* del microprocesador se conecta a un pulso de la tarjeta, en caso en que se quiera reiniciar la aplicación en ejecución.
- La salida del microprocesador *out_port* es la que contiene el resultado de los procesos realizados y es la que envía para la entrada del módulo, en este caso hacia la otra tarjeta.

[22] En el capítulo 2 se ha explicado el software empleado en el diseño de una aplicación.

Estas interconexiones se observan de una manera esquemática por medio de ISE y es la siguiente:

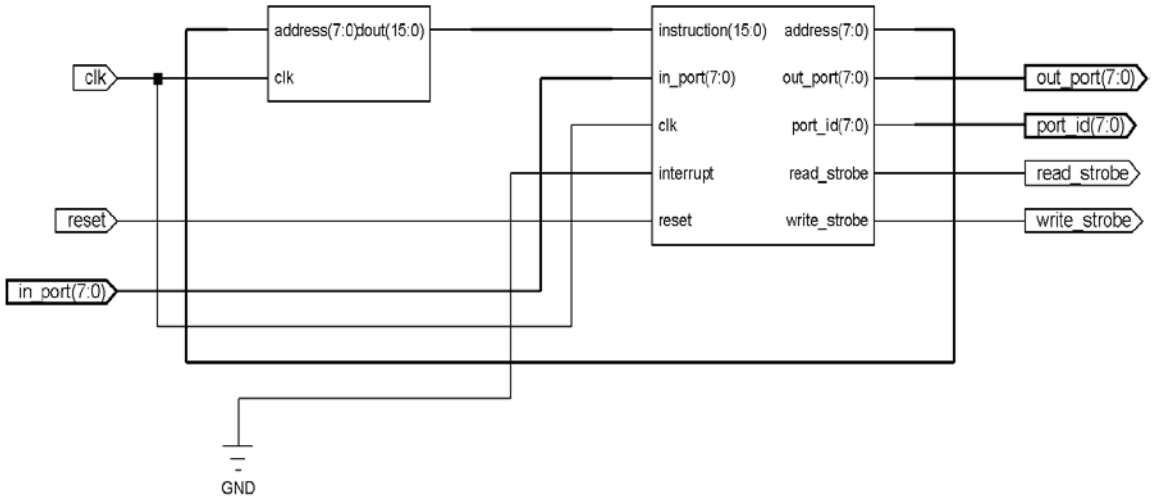

Figura 12. Interconexión de la memoria con el procesador.

Luego de observar que las conexiones arrojadas en el esquemático sean las correctas, se simula para comprobar que las operaciones realizadas por el microprocesador sea el deseado, según la aplicación implementada. Cuando se ha establecido el correcto funcionamiento se toma de base para los proyectos que se realicen.

Posteriormente se realiza un estudio del periférico a utilizar, estableciendo las señales necesarias para su funcionamiento. Luego se realiza la descripción del módulo en VHDL, se analiza el diagrama de bloques arrojado por ISE y se simula para ser implementado en la plataforma.

3.3 Ejemplo para ilustrar la implementación de un proyecto

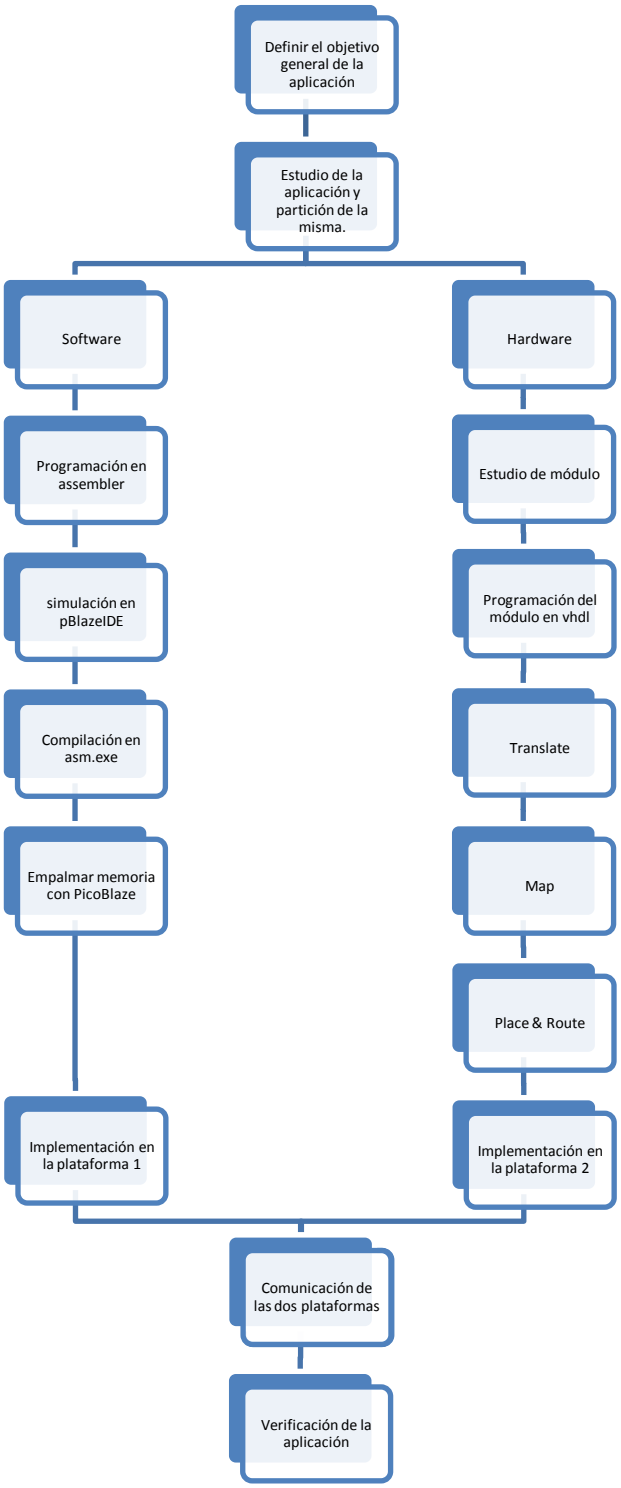

Figura 13. Diagrama de bloques de una aplicación.

Para implementar un proyecto, por ejemplo el manejo de un motor de DC, lo primero es empezar con el estudio del periférico para determinar las señales que requiere para su funcionamiento y poder realizar la descripción en VHDL del módulo, llevando un procedimiento como se observa en el diagrama de bloques de la figura 13.

Luego se visualiza el diagrama de bloques arrojado por ISE, el cual se muestra a continuación:

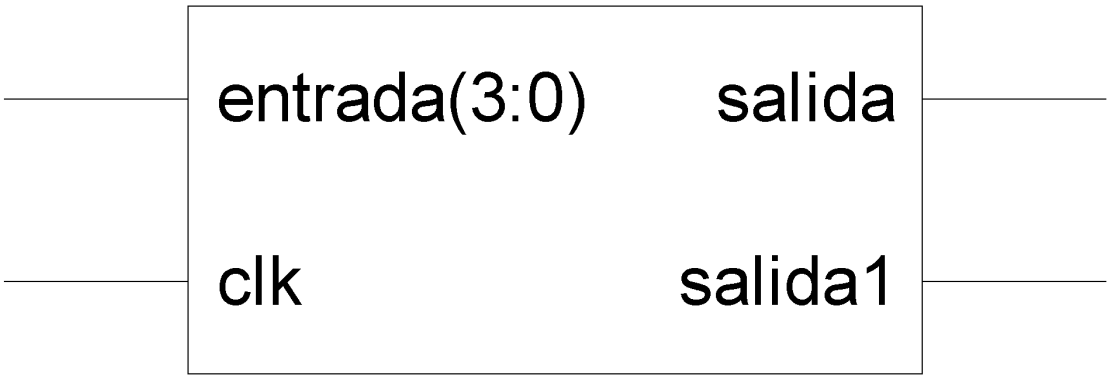

Figura 14. Diagrama de bloques de la tarjeta que contiene el periférico.

Para está tarjeta se presenta una señal de reloj de entrada, que es enviada desde la tarjeta que tiene el microprocesador para lograr sincronismo, y una señal llamada *entrada* de cuatro bits, que proviene de la señal *out_port*[23] de ocho bits[24] y de la cual se ha utilizado las cuatro bits menos significativos que son los que contienen la información de las órdenes para el manejo del periférico. Las salidas llamadas *salida y salida1,* contienen la información para el giro del motor y el PMW[25] para el manejo de la velocidad del motor, respectivamente.

[23] Revisar el capítulo 2 donde se ha explicado las señales de salida del PicoBlaze.
[24] Esta es una señal de salida del procesador y se ha explicado en el capitulo uno.
[25] Pulse Width Modulation.

Luego de la programación de la tarjeta se procede a probar el módulo diseñado y cuando se logra el correcto funcionamiento, se crea la aplicación que se quiere realizar con el periférico.

Con la aplicación idealizada se realiza la descripción en assembler en el programa pBlazeIDE, con el cual se simula para observar su correcto funcionamiento. Después de esto se compila con el ejecutable asm.exe, para obtener el programa .vhdl, luego se realiza la unión entre este archivo y el PicoBlaze.

El proyecto a nivel macro de la tarjeta que contien el PicoBlaze con la memoria, mostrado en la figura 15, presenta un bloque las señales de entrada y salida, las cuales son:

- La entrada *clk50* que es el reloj de la plataforma.
- La entrada *cs* es la proceden de los switch para el control de la velocida y dirección del motor.
- La entrada *reset* que habilita para reiniciar el microprocesador cuando se desee.
- La salida *entrada* de 8 bits que contiene la información que manejara el módulo del periférico.
- La salida *clkout* que es el reloj enviado a la segunda tarjeta.

Al analizar el contenido interno del bloque de la figura 15, se encuentra el diagrama de interconexión del procesador con su memoria y cada una de las señales que permiten esto, figura 12.

Las señales de salida de esta tarjeta son direccionadas hacia el módulo diseñado para el manejo del periférico.

Figura 15. Diagrama de bloques de la tarjeta que contiene el PicoBlaze con la memoria.

Capítulo 4

Implementación de los ES

En este capitulo se muestra el diseño de cada uno de los ES implementados. Cada ES corresponde a un módulo PMB de Digilent, el cual ofrece una serie de periféricos para ser adicionados a los sistemas de desarrollo de Xilinx. La realización de cada ES se inició con un estudio del funcionamiento de cada periférico de Digilent, para luego hacer su respectiva descripción y simulación en VHDL. Después de comprobar el correcto funcionamiento del módulo, se propuso una aplicación en la cual se requiera tanto de una parte de software y un módulo en hardware, la unión tanto del software como del hardware se hace en la parte final de cada implementación y se comprueba su correcto funcionamiento.

4.1 Módulo de salida de colector abierto: PmodOC1

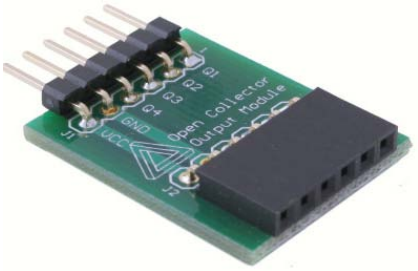

Figura 16. Módulo de salida de colector abierto [46].

4.1.1 Introducción

En las etapas con salida en colector abierto, como su nombre lo indica, la salida será directamente el colector del transistor de salida. Como se puede apreciar en la figura 17, el transistor no tiene conexión a la tensión de alimentación, por lo que se necesita para su correcto funcionamiento una conexión externa a la tensión de alimentación a través de una carga (resistencia, motor, bombillo).

4.1.2 Periférico

Este módulo puede conducir corrientes hasta de 200 mA, ya que cuenta con el MMBT3904, el cual es un transistor de salida, y es manejado por señales digitales desde el CPLD, o cualquier sistema de control digital [46]. Este transistor trabaja como un interruptor y puede manejar las corrientes para leds, motores, bombillos y otros dispositivos [46]. A continuación se muestra la estructura interna del periférico.

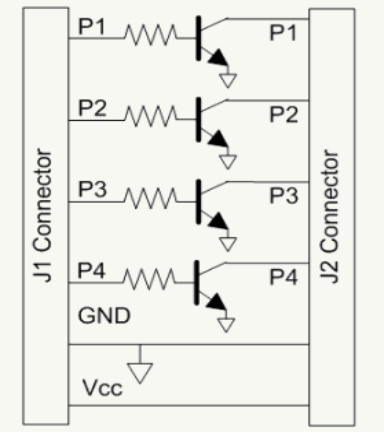

Figura 17. Circuito esquemático del módulo de salida colector abierto [46].

Las principales características del periférico son [46]:

1. Cuatro transistores de MMBT3904 de 100 mA. (máximo 200 mA).
2. Cuatro diodos de protección para los transistores en el manejo de las corrientes.
3. Conector de 6 pines para cada una de las cargas y alimentación si es necesario.
4. Un voltaje umbral de 40 voltios.
5. Un tamaño de módulo de 0.75” x 0.80”.

Cada uno de estos cuatro transistores puede funcionar independientemente de los otros, además pueden ser utilizados individualmente o simultáneamente, según lo requiera el diseño del usuario.

4.1.3 Aplicación

La aplicación diseñada para el módulo de salida colector abierto consiste en encender cuatro leds de diferentes maneras según la entrada. En caso de poner un uno lógico en el primer interruptor, los leds se encienden de uno en uno de manera horizontal hasta llegar al último y se devuelven. Cuando se mantiene en uno lógico el segundo interruptor, los leds se encienden y apagan al mismo tiempo. Cuando los dos interruptores se encuentran en un uno lógico, los leds se encienden de manera alterna, y en caso de estar los interruptores en un cero

lógico, los leds se encuentran apagados, la implementación de este periférico se presenta en la siguiente figura.

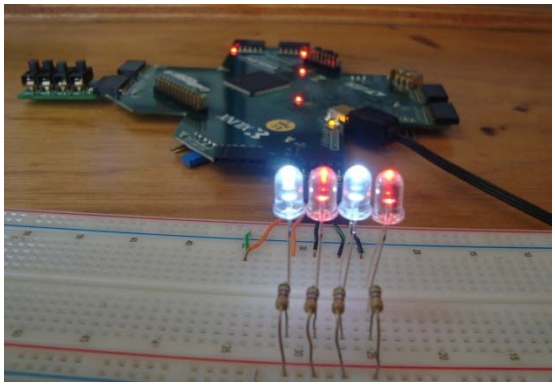

Figura 18. Implementación del ES colector abierto.

El código en assembler empleado para el diseño de la aplicación del periférico de salida de colector abierto, se encuentra en el anexo A.

4.2 Módulo para el manejo de servomotores: PmodCON3

Figura 19. Módulo para el manejo de servomotores [39].

4.2.1 Introducción

Los servomotores son los motores idóneos para aplicaciones en los que se necesita exactitud en el giro, ya que, dependiendo del ciclo útil del PMW que se envié, gira a un ángulo deseado. El servo está conformado por motor de corriente continua, una etapa de reducción y un circuito de control que cuenta con un potenciómetro que esta censando constantemente la entrada para lograr movimientos exactos y en el momento de llegar al ángulo deseado el motor se apaga, logrando un bajo consumo de potencia. El eje de un servomotor puede en algunos casos girar hasta 210 grados, pero dependiendo del fabricante este rango varia [49] [38]. La estructura interna de un servomotor es la siguiente:

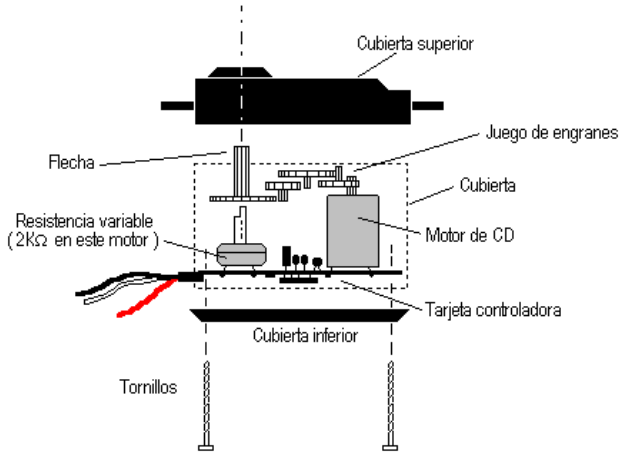

Figura 20. Estructura interna de un servomotor [49].

El servomotor utilizado en esta aplicación cuenta con tres entradas, en donde dos son de alimentación y la otra es de control. El control es realizado por medio de la modulación por anchura de pulso, PWM. Este sistema consiste en generar una onda cuadrada en la que se varía el tiempo que el pulso está a nivel alto, manteniendo el mismo período para evitar que el motor quede vibrando; con el

objetivo de modificar la posición del servo según se desee [37] [38]. Un ejemplo de PMW es mostrado a continuación:

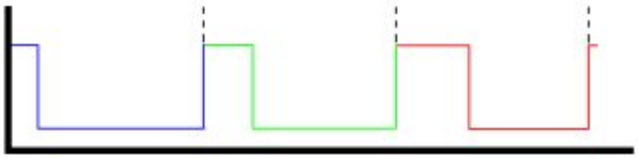

Figura 21. PWM para recorrer todo el rango de operación del servo [48].

Por medio de la variación del ancho del pulso de alto, se obtienen el movimiento deseado. El servo trabajo en un rango de 0.8 ms y 2 ms para lograr un movimiento de 0º a 180º; en caso de querer un movimiento de 0º a 90º se establece un tiempo entre 1.5 ms a 2 ms y un tiempo de bajo de 20 ms, el cual siempre se tiene que conservar para lograr movimientos exactos, ver figura 22 y 23 [48] [49] [35].

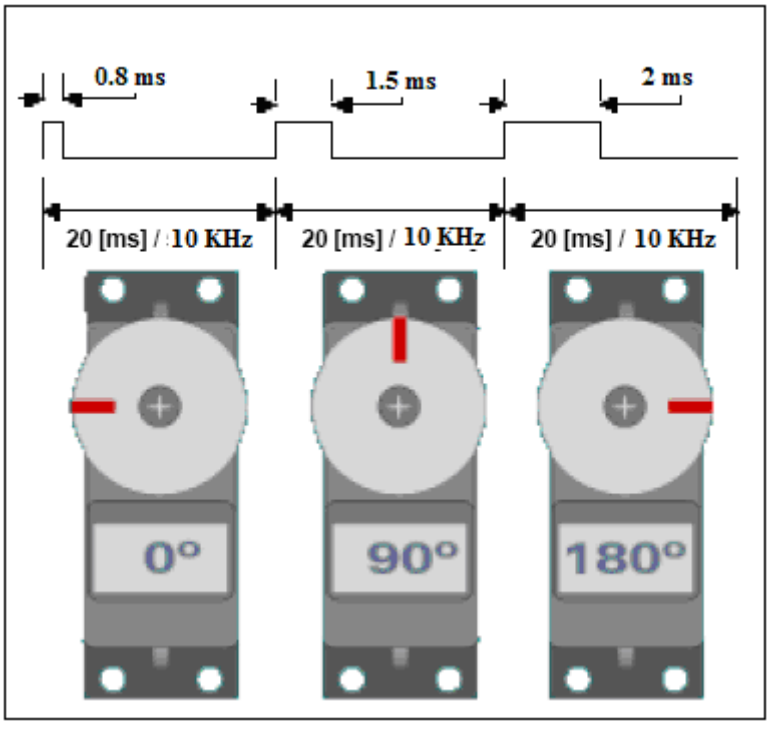

Figura 22. Ejemplos de posicionamiento de un servo [49] [35].

Para lograr un movimiento exacto del servo el periodo tiene que estar entre 19 y 21 ms, ya que si es menor de 19 ms provoca que el motor produciendo un zumbido y si es mayor de 21 ms provoca pequeños movimientos en el eje del motor [49][35].

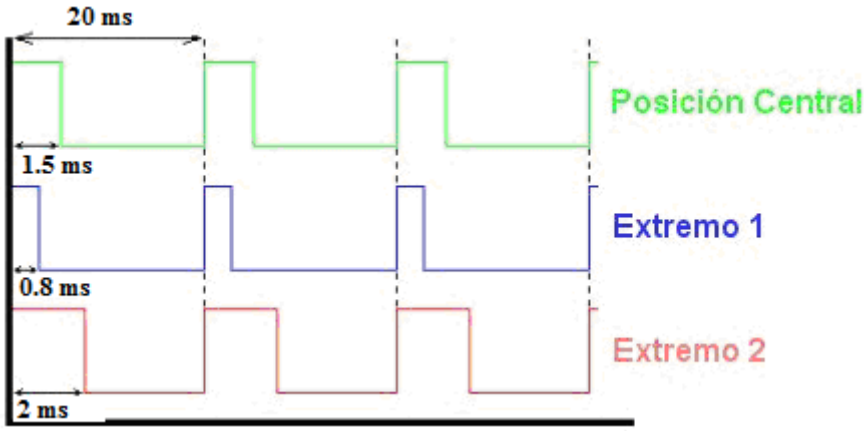

Figura 23. Periodos entre pulsos [48] [35] [38].

4.2.2 Periférico

El periférico para el manejo de servomotores cuenta con las siguientes cualidades [39]:

- Manejo de 4 servomotores de manera individual o en grupo, según se desee.
- Alimentación de los servomotores de 4.5 V a 6 V. El nivel de alimentación afectara la velocidad de posición al nuevo ángulo.
- Tamaño reducido.
- Excelente calidad.

- Cada borne para el manejo del servo cuenta con un pin de alimentación, uno de tierra y un pin de control que será empleado para enviar el PWM.

El esquema del periférico es el siguiente:

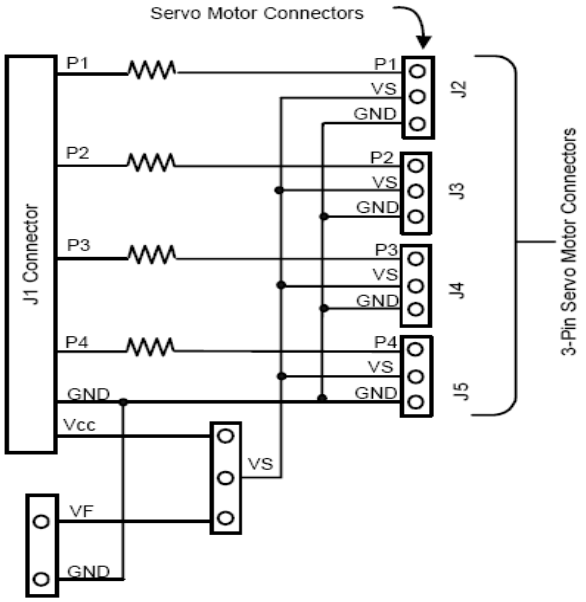

Figura 24. Esquema del periférico para el manejo de servomotores [39].

Analizando el esquema se puede observa que la única señal que se necesita enviar es la del PWM, con la cual se varia el ángulo de giro del servomotor [39].

El módulo del periférico diseñado es básicamente un PWM, en donde dependiendo de la entrada va a generar en el servomotor un movimiento de 0º a 90º ó de 0º a 180º.

El módulo cuenta con dos señales de entrada, una el reloj (clk) de frecuencia de 10 KHz y una segunda procedente del procesador (entrada), la cual nos va a indicar el tipo de PWM se va a enviar al servomotor. La señal de entrada/salida pulso es empleada para controlar la formación del PWM y la señal de salida del

módulo es pwm_out, la cual lleva el PWM que se requiere para que el motor gire el ángulo requerido.

El código en VHDL empleado en el diseño del módulo del periférico para el manejo de servomotores, se encuentra en el anexo B.

4.2.3 Aplicación

La aplicación se realizó para que el servomotor tuviese dos movimientos que dependen de la entrada. Si los interruptores están en cero, el servo está apagado; si se pone el primer interruptor en uno, el servo empezará a realizar un movimiento de 0 grados a 90 grados en un intervalo de tiempo; y si se pone el segundo interruptor en uno, el servo realizará un movimiento de 0 grados a 180 grados. En caso de poner otra entrada en los interruptores el motor estar apagado. El montaje realizado para esta aplicación se muestra a continuación.

Figura 25. Implementación del ES manejo de servomotor.

El código en assembler empleado en el diseño de la aplicación del periférico para el manejo de servomotores, se encuentra en el anexo B.

4.3 Módulo de conversor analógico / digital: Pmodad1

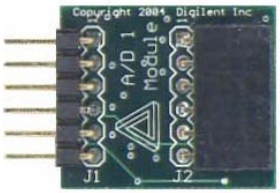

Figura 26. Módulo conversor A/D [44].

4.3.1 Introducción

La información que ofrece la naturaleza es analógica y hoy en día la mayoría de los procesadores de información son digitales, razón por la cual los conversores A/D se han convertido en una herramienta importante a la hora de procesar la información proveniente de la naturaleza [40][41].

Al convertir la señal analógica a digital se logra tener de manera sencilla el comportamiento de la naturaleza, ya que la señal digital va a tener unos valores predeterminados en un tiempo discreto y con ellas es más fácil realizar cualquier tipo de operación utilizando la inmensidad de procesadores digitales que existen en la actualidad, entre los que se encuentran los CPLDs y los FPGAs, que cada vez son más usados en procesos DSP[26]. Una de las ventajas de manejar señales digitales es el tratamiento del ruido y otras interferencias a las cuales son más sensibles las señales analógicas [40][41].

El diagrama de bloques de un conversor A/D es:

[26] Procesamiento digital de señales.

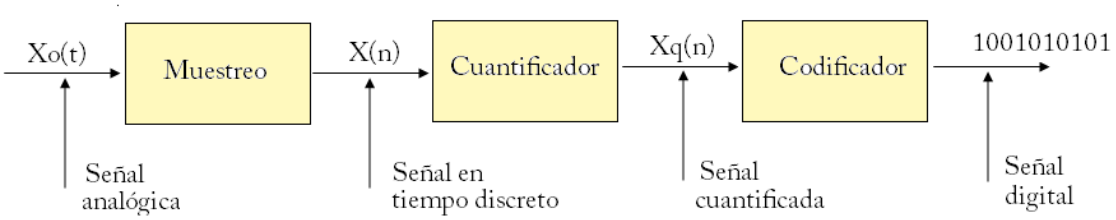

Figura 27. Diagrama de bloques de un conversor A/D.

Como se observa, la señal análoga al pasar por el muestreador se discretiza, la frecuencia del muestreador establece el tamaño de la señal discreta y del error en el momento de reconstruir la señal análoga original. Cabe recordar que esta frecuencia de muestreo tiene que ser al menos el doble de la frecuencia a reconstruir [40][41].

En el proceso de cuantificación se establece el valor de cada nivel de voltaje, que tiene cada una de las muestras tomadas y en el codificador se pasa a binario la cuantificación realizada de la señal [40].

Algunas características de los convertidores A/D son:

- Resolución: “indica el número de valores discretos que este puede producir sobre un rango de valores de voltaje, expresado en bits. La resolución puede también ser definida eléctricamente y expresarse en volts. La resolución de voltaje de un conversor es equivalente a su rango total de medida de voltaje dividido el número de valores discretos.” [41]

- Linealidad: establece que el valor análogo que está entrando tendrá su equivalencia en digital; esta característica se comprueba cuando se aplican cero voltios y en la salida digital tenemos cero.

- Precisión: es la comparación de la salida esperada y la arrojada por el conversor, determinándose el error máximo esperado [40].

- Sensibilidad: es la capacidad de efectuar conversiones para pequeñas variaciones en la entrada.
- El error de cuantificación: es la diferencia entre la señal de entrada (sin cuantificar) y la señal de salida (ya cuantificada), interesa que el ruido sea lo más bajo posible [44].
- Tiempo de conversión: es el tiempo requerido por el conversor para entregar el valor digital equivalente a la entrada analógica [42].

4.3.2 Periférico

El periférico recibe una señal análoga entre 0 [V] y 3.3 [V], la cual pasa por un filtro anti-alias que tiene dos polos en -500 Hz y 500 Hz, respectivamente. Estos filtros limitan el ancho de banda de la señal análoga para ser apropiadas para la rata de conversión [44]. Luego de ser filtrada la señal, se pasa por el conversor y es convertida a una señal digital de 12 bit en un rango de 0 a 4095 [44].

El periférico utiliza un bus serial para la transmisión de los datos a una frecuencia de 20 MHz, el cual emplea un protocolo de comunicación SPI/MICROWIRE[27].

El periférico del conversor A/D es:

[27] El SPI es un estándar muy suelto para controlar casi cualquier electrónica digital que acepte un reloj y el Microwire es un subconjunto de la SPI.

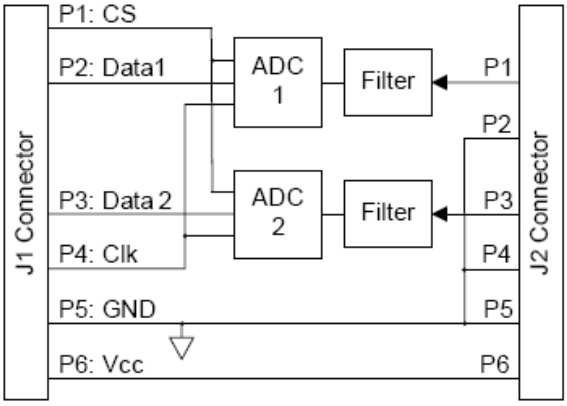

Figura 28. Esquema del conversor A/D [35].

Algunas características del periférico son [44]:

- La rata de conversión es de un millón de muestras por segundo.
- Dos conversores ADCS7476MSPS de 12-bit.
- Dos filtros para anti-alias.
- Dos canales simultáneos para conversión.
- Bajo consumo de potencia.
- Portable.

El módulo para el manejo del periférico se diseño teniendo en cuenta las características de funcionamiento del conversor ADCS7476MSPS, el cual tiene una bandera de habilitación para entregar los datos digitales, los cuales vienen en un vector de 16 datos ordenados de la siguiente manera, los cuatro más significativos son cero y los demás son los datos de la conversión, ver figura 29 [44][45].

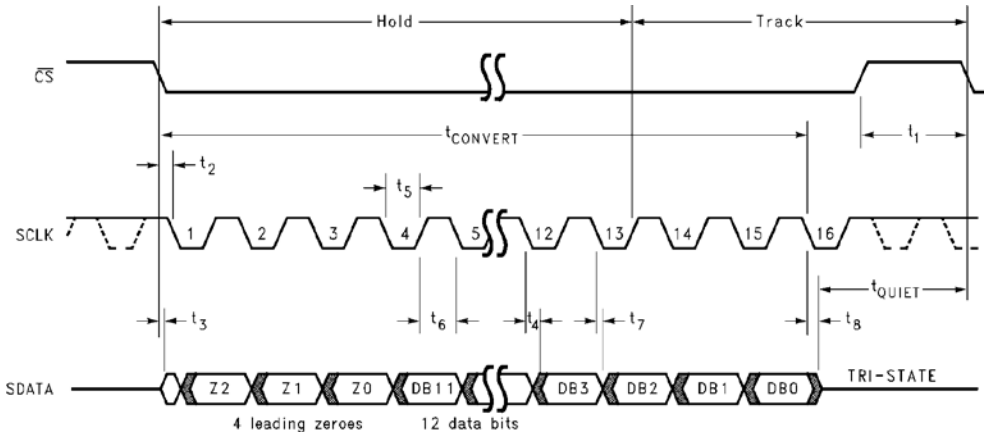

Figura 29. Diagrama para la conversión de datos [45].

Teniendo presente estas características, el módulo para el control del periférico visualiza los datos convertidos en los leds de la tarjeta y en el display siete segmentos, y estos datos también son enviados al procesador para que este los procese y active una alarma.

El código en VHDL empleado en el diseño del módulo del periférico para el manejo del conversor analógico/digital, se encuentra en el anexo C.

4.3.3 Aplicación

Para realizar un proceso de control, generalmente es necesario digitalizar señales provenientes del medio a controlar y verificar que estas señales se mantengan en algunos niveles pre-establecidos. En este caso queremos simular la señal analógica, digitalizarla y mostrar su valor digitalizado en un display siete segmentos, de igual manera y se estableció un rango voltaje de verificación, para ser observado en una alarma visual.

La aplicación establecida con el PicoBlaze, es detectar los rangos establecidos para la alarma y mostrar en dos leds el momento en que se encuentra la entrada dentro de estos rangos. El montaje realizado para esta aplicación es:

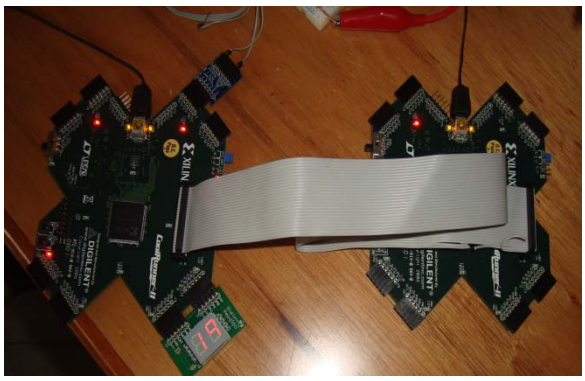

Figura 30. Implementación del ES conversor analógica digital.

El código en assembler empleado en el diseño de la aplicación del periférico para el manejo del conversor analógico/digital, se encuentra en el anexo C.

4.4 Módulo convertidor digital / analógico: PModDA2

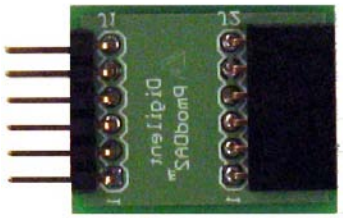

Figura 31. Módulo del convertidor D/A [36].

4.4.1 Introducción

Los conversores D/A fueron empleados al presentarse la necesidad de recuperar la señal analógica, después de que está fuera tratada o transmitida. Es así como los conversores D/A se convirtieron en la herramienta que permitía a los sistemas la comunicación con el medio y de esta manera ejercer algún tipo de control [46].

Los conversores D/A se utilizan en los reproductores de discos compactos, cintas de vídeos digitales y en los equipos de procesamiento de señales digitales de sonido y vídeo, entre otras aplicaciones [46].

La mayoría de los conversores D/A emplean una red similar a una red reostática[28], por la cual pasan los datos digitales en paquete de bits. Las resistencias varia proporcionalmente con el valor binario recibido [46] [47] [48].

El diagrama de bloques de un convertidor D/A es:

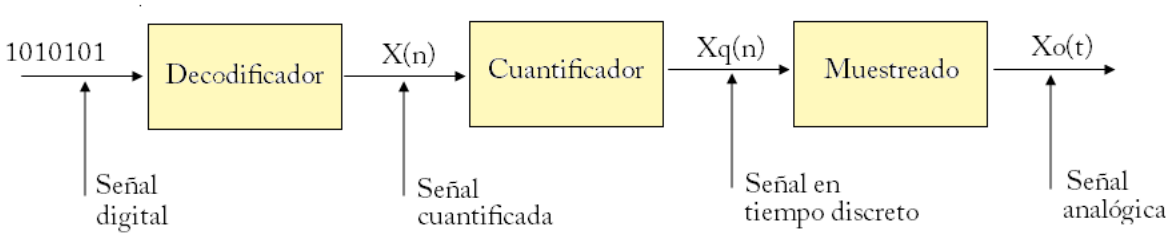

Figura 32. Diagrama de bloques de un convertidor D/A.

[28] Componente eléctrico para regular la intensidad de la corriente sin necesidad de abrir el circuito y que consiste en una resistencia eléctrica que puede variarse a voluntad

4.4.2 Periférico

El periférico PmodDA2 convierte señales digitales a valores analógicos de voltaje, por medio de dos canales simultáneos con una resolución de 12 bits. El periférico puede producir una señal de salida análoga en un rango de 0 [V] a 3.3 [V], la cual es obtenida de una señal digital de entrada de 12 bits [36].

El periférico emplea un bus serial para la transmisión de datos, el cual es SPI/MICROWIRE, con una frecuencia de 20 MHz. Los dos conversores están conectados en paralelo para poder trabajar con los dos canales de conversión de forma simultánea [36].

El periférico puede ser alimentado entre 2.7 [V] y 5.5 [V], pero es recomendable que este valor sea 3.3 [V] para evitar daños en el periférico y en la tarjeta [36].

La estructura del periférico de convertidor D/A es:

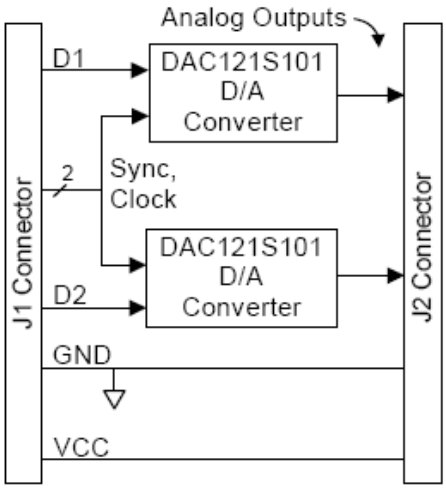

Figura 33. Esquema del periférico D/A [36].

Algunas características del periférico son [36]:

- Dos conversores DAC121S101 de 12 bits.
- Dos canales para conversiones simultáneas.
- Bajo consumo de potencia.
- Portable.

El convertidor DAC121S101 con que cuenta el periférico convierte señales de 12 bits en un voltaje de salida, este puede operar con voltaje de 2.7 [V] a 5.5 [V] y consume 177 µA para 3.6 [V]. Este convertidor funciona con una frecuencia máxima de 30 MHz y es compatible con interfaz SPI. Este convertidor emplea una bandera para recibir los datos a ser convertidos, de los 12 datos de entrada los cuatro más significativos son cero y los siguientes son los datos a convertir, ver figura 34 [35].

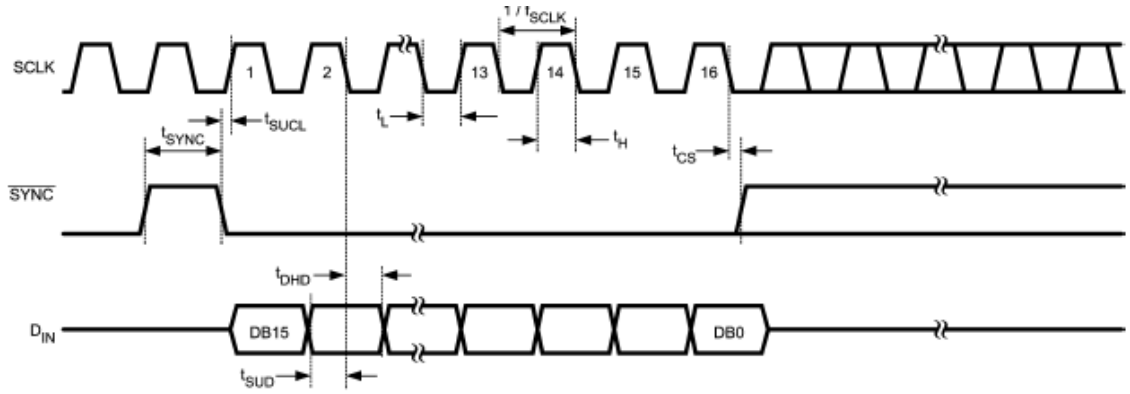

Figura 34. Diagrama para la conversión de datos [35].

Después de analizadas las características del periférico se diseño el módulo, el cual recibe los datos digitales para ser convertidos y le envía secuencialmente estos datos al periférico.

El código en VHDL empleado en el diseño del módulo del periférico para el manejo del conversor digital/analógico, se encuentra en el anexo D.

4.4.3 Aplicación

En la industrial es común encontrar conversores D/A para visualizar el comportamiento de un instrumento o para lograr el control de un sistema. La aplicación diseñada se enfoco en la construcción de graficas tipo rampa, cuadrada y sinusoidal, según se programe el procesador. El procesador envía secuencialmente cada uno de los datos esperando la bandera de habilitación que es enviada desde el convertidor.

El montaje realizado para esta aplicación se muestra a continuación.

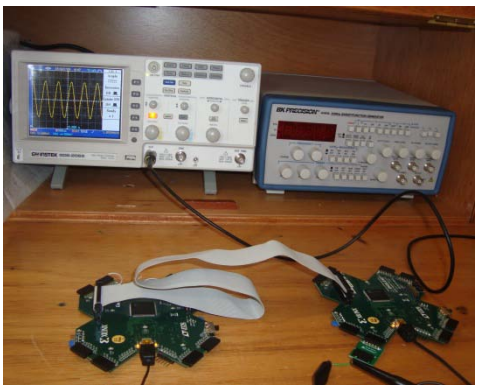

Figura 35. Implementación del ES digital analógico.

El código en assembler empleado en el diseño de la aplicación del periférico para el manejo del conversor digital/analógico, se encuentra en el anexo D.

4.5 Módulo del puente h: PmodHB3.

Figura 36. Módulo del puente H [37].

4.5.1 Introducción

El puente H en una interfaz de potencia que permite el manejo de motores DC por medio de señales de baja potencia, que provienen de un circuito [37]. Algunas características de la interfaz puente H son [37]:

- Manejo de señales de baja potencia: antes de hacer la conexión de un motor es aconsejable en mirar el consumo del motor para evitar dañar el puente H.
- Capacidad para mover el motor en los sentidos: esto se logra con el envió de señales lógicas que van a provocar la inversión en la polaridad en la alimentación del motor.
- Practicidad en la conexión por sus borneras.
- Portable.

El esquema tipo de un puente H es:

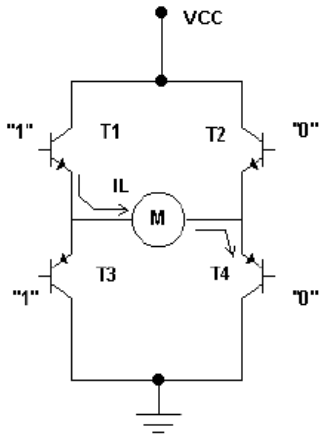

Figura 37. Esquema del puente H [48].

Como se observa, según el valor lógico se coloque en las entradas de los transistores va a fluir la corriente y de esta manera se logra el giro del motor deseado [38].

4.5.2 Periférico

El periférico es el módulo del puente H para el manejo de motores DC y algunas características son [37]:

1. La corriente máxima para PmodHB3 es de 2 amperios.
2. La alimentación del puente H puede estar entre 2.5 y 5 voltios, pero normalmente es manejado a 3.3 voltios, al ser conectado con la tarjeta de desarrollo de CPLD.
3. El PmodHB3 está en condiciones de manejar motores de DC hasta de 12 voltios.

4. Es recomendable manejar la velocidad por una señal PWM (Modulación por ancho de pulsos).
5. No es recomendable realizar un cambio de dirección a altas velocidades para esto es necesario detener el motor.

Los componentes del periférico del puente H son [37]

1. Bornera para alimentación: A esta bornera se acopla la fuente de alimentación de la cual el motor a controlar drenará la potencia necesaria para su operación, es importante realizar la conexión con la polaridad indicada en el módulo y se debe estar seguro de que esta fuente tiene capacidad para suministrar con seguridad la corriente requerida por el motor, en el módulo está representado por VM y GND.

2. Entrada de control: A esta entrada se acopla el sistema digital con el cual se desea controlar el motor, se debe tener en cuenta que una de las señales es la que indica la dirección del motor y la otra señal habilita la conducción de la corriente para el motor.

3. Bornera para conexión del motor: En esta bornera se acopla el motor, en el módulo se representa como M+ y M-.

4. Transistores de potencia: Es importante considerar la importancia de estos dispositivos pues en caso de controlar motores que consuman elevadas corrientes con respecto a la corriente permitida por el módulo (2 A, para una alimentación de motor 12 Voltios DC).

En la figura 38 se muestra la estructura interna del periférico.

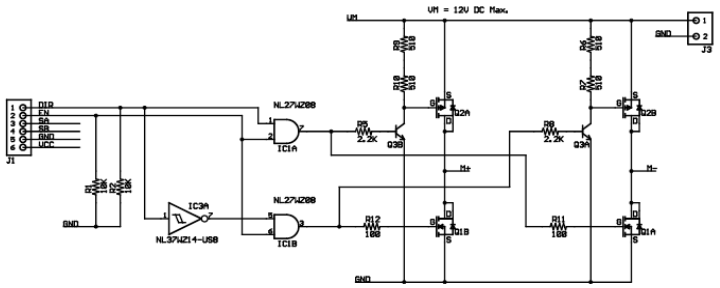

Figura 38. Esquemático de PmodHB3 de Digilent [37].

Al analizar el esquema del módulo se observa como las señales digitales activan o desactivan un lazo para la conducción de la corriente del motor de DC, la dirección se encarga de activar uno de los dos lazos según sea su nivel, cuando sea un alto es un lazo y cuando sea un bajo será el otro la lazo, el la señal EN (habilitador) se encarga de cerrar o abrir el lazo según el usuario desee [37]. Una forma de control de velocidad de un motor de DC es la modulación por ancho de pulsos (PWM), por tal razón el diseño del módulo se basa en este principio.

El módulo cuenta con dos señales de entrada, una de ellas con un bit llamado clk, la cual es la señal de reloj del sistema, con una frecuencia de 10 KHz. La segunda señal (dir_vel de dos bit) es enviada por el procesador la cual le indica al módulo la modulación del PWM y la dirección con la cual debe girar el motor (el bit menos significativo indica la dirección del motor y el bit mas significativo nos indica la velocidad). Las señales de salidas son las que controlaran el periférico puente. Como los nombres de las señales lo indica, **dir** será la señal que indicara la dirección del motor y la señal **pwm_out** indica la señal PWM que activara la conducción de corriente para el motor.

El código en VHDL empleado en el diseño del módulo del periférico para el manejo del puente H, se encuentra en el anexo E.

4.5.3 Aplicación

En la aplicación diseñada el usuario tiene la opción de escoger entre dos velocidades de giro de motor, en el que el procesador recibe dos entradas, las cuales son comparadas y dependiendo de esta comparación el programa es enviado a unas subrutinas las cuales indican al módulo el movimiento del motor, una de esta rutinas es hacer el motor en un sentido, luego se detendrá y cambiara de sentido y así sucesivamente con una baja velocidad, la segunda rutina será semejante pero con mayor velocidad y también hay un estado donde el motor no tendrá movimiento según las señales de entrada. El montaje realizado para esta aplicación es:

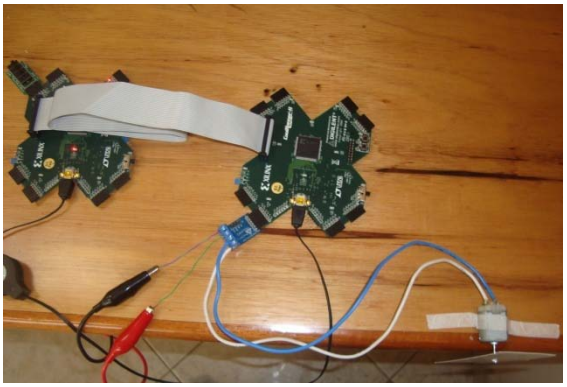

Figura 39. Implementación ES puente H.

El código en assembler empleado en el diseño de la aplicación del periférico para el manejo del puente H,se encuentra en el anexo E.

4.6 Módulo amplificador de audio: PmodAMP1

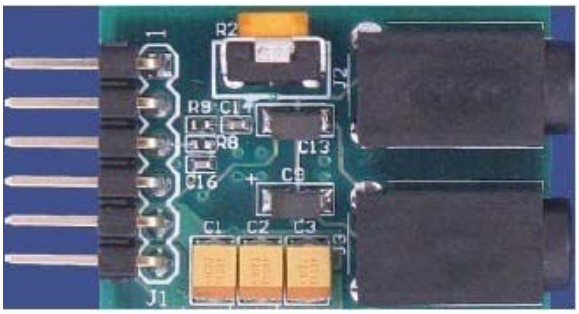

Figura 40. Módulo del amplificador de audio [40].

4.6.1 Introducción

Los amplificadores operacionales son circuitos que se emplean para obtener una salida amplificada en un valor, respecto a la entrada [42]. Los amplificadores cuentan con restricciones las cuales afectan la salida que se espera, ya que no puede generar una amplificación muy grande porque se puede obtener una salida distorsionada, y no se puede tener un nivel de voltaje en la salida mayor que el de alimentación porque podría dañar el amplificador [42].

Aunque los amplificadores tengan una ganancia Av, esto no es lo único que se tiene que tener en cuenta para conectarlo al altavoz, sino la corriente que es capaz de suministrar [43]. Es por esta razón que es necesario poner en cascada con el amplificador operacional un transistor, que sea capaz de suministrar la corriente necesario para que en el altavoz su pueda escuchar música [43]. El conjunto del amplificador operacional y el transistor es lo que se llama amplificador de audio y es mostrado en la siguiente grafica.

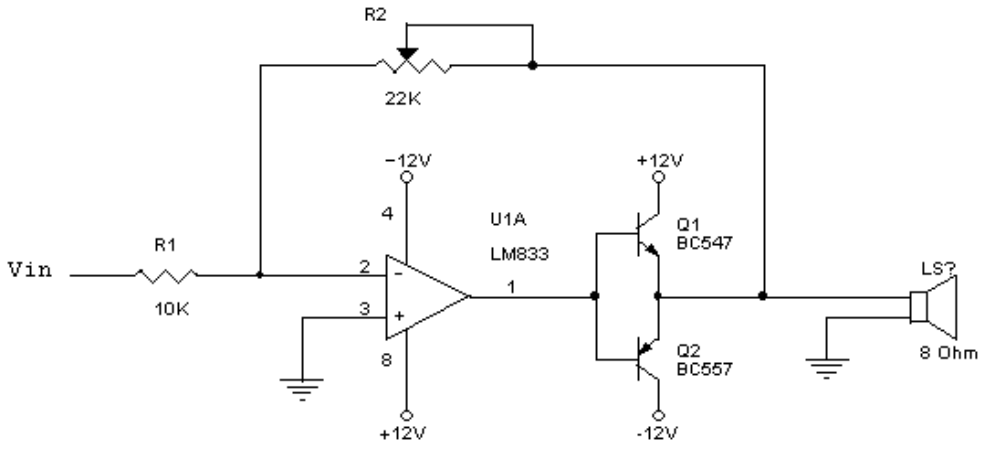

Figura 41. Esquema de un amplificador de audio [43].

En la anterior grafica se muestra el amplificador operacional en cascada con una clase AB[29] para suministrar la corriente que necesita el parlante y el volumen es controlado por medio de la resistencia variable que está formando la realimentación.

4.6.2 Periférico

El periférico es un amplificador de audio de bajas señales de audio para auriculares de equipo de música y parlantes monofónicos [41]. Algunas características del módulo son [41]:

- Cuenta con un amplificador de audio IC LM4838 de National Semiconductor.

[29] Los amplificadores están clasificados en tipos A, B, AB y C; de los cuales el tipo AB es el empleado para los amplificadores de audio 44.

- Un conector de sonido de 1/8 de pulgada para un auricular de música.
- Un conector de sonido monofónico de 1/8 de pulgada.
- Opera con un voltaje de 3 [V] a 5 [V].
- Portable.

La estructura interna del periférico es:

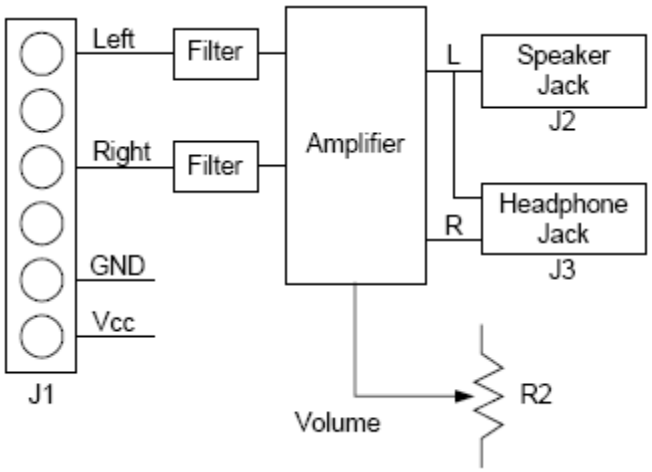

Figura 42. Estructura del módulo amplificador de audio 41.

El módulo acepta entradas digitales o análogas para ser amplificadas [41]. El módulo es alimentado por medio de la tarjeta a 3.3 [V], mediante una conexión interna. También se cuenta con un filtro pasa bandas en la entrada, con sus frecuencias localizadas en 150 [Hz] y 8 [KHz] [41].

Se podría usar un PMW producido por la tarjeta como entrada digital [41]. El filtro pasa bajas en la entrada actuaria como un filtro reconstructor para convertir el PMW en una señal de voltaje análogo para la entrada del amplificador [41].

El módulo PmodAMP1 acepta entradas análogas con un rango de voltaje de 0 – Vcc [41]. Esta entrada análoga podría provenir de un módulo conversor analógico/digital que la tarjeta de Digilent maneje [41].

La entrada de voltaje es filtrada por el filtro pasa bajas, amplificada y luego es enviada al amplificador de audio estéreo y monofónico [41].

Teniendo en cuenta las características del módulo amplificador de audio, se consultaron las frecuencias de las notas musicales para realizar un divisor de frecuencia para obtener el tono deseado. Las frecuencias son:

	Notas musicales						
	Do	**Re**	**Mi**	**Fa**	**Sol**	**La**	**Si**
Orden	1	2	3	4	5	6	7
Frecuencia	261	294	330	349	392	440	494

Tabla 1. Asignación de frecuencia según nota musical [44].

Con la distribución de frecuencias, se creó el módulo en VHDL, el cual genera la frecuencia de la nota musical seleccionada.

El código en VHDL empleado en el diseño del módulo del periférico amplificador de audio, se encuentra en el anexo F.

4.6.3 Aplicación

La aplicación diseñada consiste en reproducir parte de la canción de la alegría, para esto el procesador envía la nota que sigue y el módulo establece la frecuencia a la que tiene que funcionar para sacar la nota. Cuando se quiere empezar con la reproducción de la canción, se cierra el switch y el procesador empieza a enviar cada nota en el orden correcto (ver tabla 1) con intervalos de tiempo, para obtener una canción semejante a la canción de la alegría. El montaje realizado para esta aplicación es:

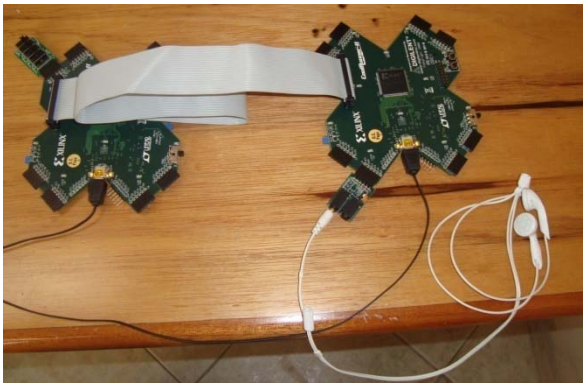

Figura 43. Implementación de la aplicación realizada para el amplificador de audio.

El código en assembler empleado en el diseño de la aplicación del periférico amplificador de audio, se encuentra en el anexo F.

4.7 Módulo para el manejo de puerto serial RS232: PmodRS232

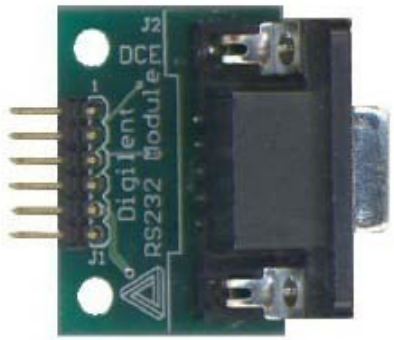

Figura 44. Módulo del puerto serial RS232 45.

4.7.1 Introducción

Desde hace varios años se ha presentado la necesidad de realizar la comunicación entre ordenadores, con otros dispositivos para ampliar los recursos a manipular, etc. Es por esto que la EIA[30] propuso la comunicación serial asíncrona, llamada puerto serie RS232, conocida a nivel mundial [46]. En la actualidad los computadores ya no traen el puerto serial, este ha sido reemplazado por el puerto USB, aunque sigue siendo una alternativa practica de comunicación, es por esto que en el mercado se puede conseguir cables de convierten de puerto USB a puerto serial RS232.

Protocolo de comunicación

Mediante esta comunicación serial se puede enviar datos hasta de 8 bits y la velocidad es medida en baudios[31] y es necesario tres líneas para la comunicación

[30] Asociación de Industrias Electrónicas

[31] Los baudios son medidos en bits/segundo y normalmente es de 9600 bits/segundo.

una de transmisión, otro de recepción y la tierra, empleada como referencia para las señales [46]. Estos datos son enviados de manera asíncrona por carácter y síncrono por bit, los bits enviados en total son [46][49]:

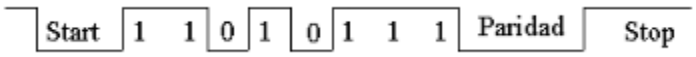

Figura 45. Esquema de los bits enviados en una comunicación [49].

- El bit de inicio: este bit vale cero lógico.
- Los datos son enviados en paquetes de 8 bits, empezando por el menos significativo y terminando con el más significativo.
- El bit de parada: este bit vale uno lógico.

Cuando se envía un dato que contiene más de ocho bits, se envía por paquetes de ochos bits cada uno por aparte manteniendo los bits de inicio y de parada.

La comunicación por medio del RS232 es full dúplex, la cual permite enviar y transmitir al mismo tiempo para distancias menores a 30 [m], desde 3 a 29 hilos [48].

Para tener dominio sobre el puerto serial la CPU emplea direcciones de entrada, salida y líneas de interrupción (IRQ), las cuales son empleados para indicar que ha ocurrido un evento, como la llegada de un dato o que algunas señales cambiaron de estado [47].

La CPU tiene una velocidad de respuesta a las interrupciones alta para recoger el dato, evitando que el próximo dato lo sobrescriba [47]. Una forma de solucionar el

problema de generar muchas interrupciones, es el empleo de las UART[32], que viene integrado en el computador y cumple con las siguientes funciones [47] [49]:

- Convierte de paralelo a serie los ocho bits que recibe del procesador y convierte las señales que recibe de la transmisión de serie a paralelo para entregárselas al procesador.
- Adiciona los bits de inicio, parada y paridad a cada carácter que va a ser transmitido y los elimina cuando recibe un dato.
- Supervisa que la velocidad de transmisión sea la correcta.
- Supervisa el bit de paridad y envía los posibles errores que se presente durante la comunicación.
- Controla el estado lógico de la señal RSLD para informar cualquier anomalía al procesador.

El protocolo de comunicación se puede realizar por [48]:

- Hardware (XON/XOFF) 48: el carácter XON es empleado por el receptor para indicar que si hay espacio para recibir los datos y cuando no tiene espacio envía el carácter XOFF.
- Software (handshaking RTS/CTS) 48: con este tipo de comunicación se emplean dos líneas de comunicación, en donde CTS es la bandera que informa que el receptor se encuentra listo para la transmisión y RTS informa que el computador se encuentra lista también para la transmisión.

Los pasos para realizar la comunicación son [46]:

[32] Universal Asynchronous Receiver/Transmitter, en español: Transmisor Receptor Asíncrono Universal.

- Configuración de la velocidad de transmisión: esta viene por defecto establecida en 9600 bits/segundo.
- Protocolo serie.
- Protocolo de control de flujo.

4.7.2 Periférico

El periférico es un módulo para el manejo de un puerto RS232, el cual convierte niveles de voltaje desde niveles lógicos usados por la tarjeta a voltajes usados para la comunicación serie del RS232, los cuales son: para un uno lógico el voltaje requerido se encuentra entre -3 y -12 V, y para un cero lógico el voltaje requerido se encuentra entre 3 y 12V [45]. Algunas características del periférico son:

- Trae integrado un Max3223.
- Un conector DB9.
- Comunicación full dúplex.
- Portable.

La estructura del periférico es:

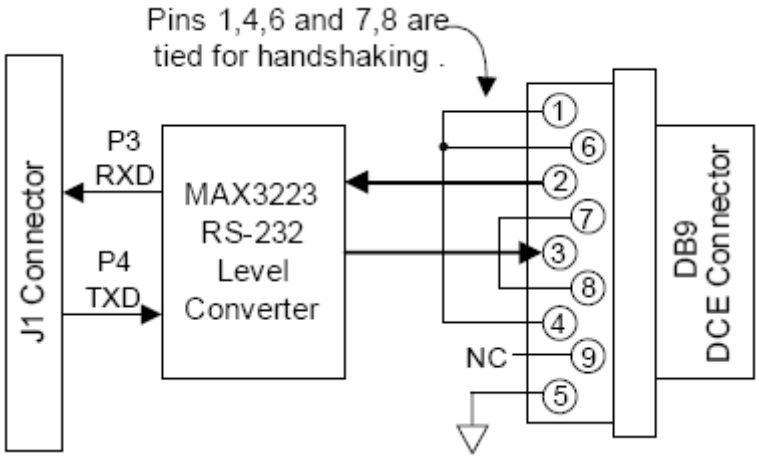

Figura 46. Estructura del módulo para el manejo del puerto serie RS232 [48].

El módulo del periférico está configurado con tres líneas de comunicación, una de transmisión, una de recepción y la tierra [48].

El módulo está diseñado para el manejo de 8 Bits de datos, por tal razón es necesario utilizar dos señales para la comunicación entre el CPLD y el dispositivo, una como entrada la cual se denomina *entrada*, esta señal es de 1 bit debido a que el dato es recibido de forma serial y es recibida del dispositivo con el cual se esté realizando la comunicación. La otra señal empleada es la de salida, denominada *salida_t*, la cual es enviada de forma serial desde el CPLD, pasando por el periférico y llegando al dispositivo con el que se establece la comunicación.

El usuario decide el dato a transmitir desde el CPLD al dispositivo, este dato es recibido por el CPLD, de forma paralela y el nombre de la señal es *entrada_t* de 8 bits. Al enviar este dato al dispositivo, se tendrá un resultado o una respuesta, es observar por medio de 8 leds de la plataforma y la señal se denominada mostrar de 8 bits.

Para control de la recepción y transmisión de datos se empleó una bandera llamada *envio_t*, cuando se encuentre en nivel bajo, el módulo estará recibiendo el dato del usuario, y en nivel alto el dato recibido será enviado al dispositivo.

En este protocolo de comunicación se establece una velocidad de 9600 Baudios, no se utiliza bit de paridad y el tamaño del dato es de 8 Bits y los buffer son de un dato. Para la recepción del dato fue necesario realizar un sobremuestreo para evitar errores, por tal razón se empleo un reloj de 100 KHz y divisor de frecuencia para trabajar a 9600 Baudios en transmisión.

El código en VHDL empleado en el diseño del módulo del periférico para el manejo de puerto serial RS232, se encuentra en el anexo G.

4.7.3 Aplicación

La aplicación consiste en que el usuario envía un dato a Matlab, en donde realiza una operación, en este caso división, y envía el resultado a la plataforma, la cual lo muestra al usuario por medio de los leds.

El microprocesador es empleado como medio de comunicación entre el usuario y el módulo para el manejo del periférico, el código utilizado en assembler se encuentra en el anexo G.

Figura 47. Implementación del ES del RS232.

El código utilizado en Matlab en el diseño de la aplicación del periférico para el manejo de puerto serial RS232, se encuentra en el anexo G.

Conclusiones

- La implementación del microcontrolador PicoBlaze (soft-core) de Xilinx, sobre la plataforma CoolRunner II permite generar *embedded systems* que consten tanto de una parte de software como de hardware.

- El uso de periféricos externos como los PMB de Digilent, permiten darle una mayor flexibilidad y versatilidad al diseño de ES.

- La unión entre software y hardware para el desarrollo de un *embedded system*, permite generar sistemas de mayor rendimiento y con la posibilidad de implementar aplicaciones cada vez más robustas.

- Una alternativa a la hora ES de mayor necesidad de recursos es la utilización de varios CPLDs interconectados entre sí.

Referencias bibliográficas

1. Hoja de datos del CPLD XC2C256 de la CoolRunner - II. 2007, Vol. v3.2.
2. http://perso.wanadoo.es/pictob/imagenes/pld.gif. [En línea] [Citado el: 10 de diciembre de 2008.]
3. http://www.elprisma.com/apuntes/ingenieria_industrial/disenoasistidoporcomputadora/default.asp. [En línea] [Citado el: 04 de septiembre de 2009.]
4. http://www.larevistainformatica.com/DISENO-ASISTIDO-COMPUTADORA.HTML. [En línea] [Citado el: 04 de septiembre de 2009.]
5. Introducción al software Xilinx. s.l. : Universidad de Valencia, Vol. v 6.
6. Manual de referencia de la tarjeta CoolRunner - II de Xilinx. 2007.
7. http://www.alegsa.com.ar/Dic/cmos.php. [En línea] [Citado el: 04 de septiembre de 2009.]
8. http://www.xilinx.com/itp/xilinx4/pdf/docs/hdb/hdb.pdf. [En línea] [Citado el: 04 de septiembre de 2009.]

9 http://www.clarin.com/suplementos/informatica/1999/06/23/t-02802i.htm. Revisado diciembre de 2008.

10 SALAMANCA B., William; CARRILLO H., Oscar H.; “Diseño de una plataforma para el desarrollo de embedded systems ”; Proyecto de pregrado; UIS, 2007.

11 Guía de usuario para el microcontrolador embedded PicoBlaze 8-bit para CPLDs.

12 http://iie.fing.edu.uy/jornadasISTEC/presentaciones/excalibur/a4.pdf. Revisado mayo de 2009.

13 FERNÁNDEZ C., José Carlos; “Estrategias para el desarrollo de aceleradores hardware de algoritmos basados en Redes Neuronales”; Proyecto de fin de carrera; Universidad Politécnica de Cartagena, febrero de 2006.

14 MARTINEZ, B.; TARRUELLA, D; “Adaptación de un driver WLAN Linux a una arquitectura NIOS-Avalon”; Universidad Autónoma de Barcelona.

15 http://cephis.uab.es/resources/pdf/papers/JCRA_2006_UFCpp.pdf. Revisado mayo de 2009

16 Manual de referencia del módulo de salida de colector abierto PmodOC1 de Digilent.

17 http://www.todorobot.com.ar/documentos/servomotor.pdf. Revisado junio de 2009.

18 http://cfievalladolid2.net/tecno/cyr_01/robotica/sistema/motores_servo.htm. Revisado junio de 2009.

19 http://www.tupublicas.com/docs/02-07-2009-26-.doc. Revisado junio de 2009.

20 http://www.monografias.com/trabajos60/servo-motores/servo-motores2.shtml. Revisado junio de 2009.

21 http://www.webelectronica.com.ar/news35/nota03.htm. Revisado junio de 2009.

22 http://www.ime.eb.br/~pinho/micro/trabalhos/Robot_Bioins_I.pdf. Revisado junio de 2009

23 http://www.info-ab.uclm.es/labelec/Solar/electronica/elementos/servomotor.htm. Revisado junio de 2009.

24 Manual de referencia del conector para servomotores de Digilent.

25 http://www.ipls-lasalle.org/foroipls/?p=320. Revisado abril de 2009.

26 http://www.alegsa.com.ar/Dic/conversor%20analogico-digital.php. Revisado mayo de 2009.

27 http://server-die.alc.upv.es/asignaturas/PAEEES/2004-05/A02-A03%20-%20Conversor%20AD%20del%20PIC16F877.pdf. Revisado mayo de 2009.

28 http://www.i-micro.com/pdf/articulos/spi.pdf. Revisado mayo de 2009.

29 Manual de referencia del módulo conversor análogo a digital PmodAD1 de Digilent.

30 Hoja de datos del conversor análogo a digital ADCS7476.

31 www.terra.es/personal/lermon/cat/articles/evin0122.htm. Revisado mayo de 2009.

32 www.babylon.com/definition/Conversor_digital-analógico/Spanish. Revisado mayo de 2009.
33 http://148.202.12.20/~osalas/instrumentacion/DAC.htm. Revisado mayo de 2009.
34 http://www.el.uma.es/marin/Practica3.pdf. Revisado mayo de 2009.
35 Hoja de datos del conversor digital análogo DAC121S101 12-Bit.
36 Hoja de datos del módulo digital análogo PModDA2 de Diligent.
37 Manual de referencia del módulo puente H para el manejo de motores de corriente continua de Digilent.
38 JIMÉNEZ, Carlos F.; "PUENTE H interfaz de potencia para motores de DC"; guía de instalación; octubre de 2005.
39 http://www.control-gray.googlecode.com/files/PuenteH.PDF. Revisado mayo de 2009.
40 Manual de referencia del módulo amplificador de audio PmodAMP1 de Digilent.
41 http://www.unicrom.com/Tut_amplificadores_.asp. Revisado agosto de 2009.
42 http://www.mailxmail.com/curso-amplificadores-operacionales/pequeno-amplificador-audio. Revisado agosto de 2009.
43 http://www.electronica2000.com/amplificadores/amplif.htm. Revisado agosto de 2009.
44 http://webdelprofesor.ula.ve/ciencias/sanrey/tubos.pdf. Revisado agosto de 2009.
45 Manual de referencia del módulo para el manejo de puerto serial RS232 PmodRS232 de Digilent.
46 http://alcabot.org/seminario2006/Trabajos/JoseManuelMurciaBarba.pdf. Revisado agosto de 2009.
47 http://www.euskalnet.net/shizuka/rs232.htm. Revisado agosto de 2009.
48 http://rdedatos.tripod.com/rs232.htm. Revisado agosto de 2009.
49 http://www.docstoc.com/docs/377386/norma-rs232. Revisado octubre 2009
50 http://www.diccionarioweb.org/d/ES-ES/reostatica. Revisado octubre 2009

51 http://www.clarin.com/suplementos/informatica/1999/06/23/t-02802i.htm. Revisado diciembre de 2008.

52 http://www.emagister.com.mx/index_buscador.cfm?action=search&frmIdCateg=29&frmIdCentro=738069126734871776412440869143 22&ignorarCustomFilter=1. Revisado diciembre 2008.

53 http://dialnet.unirioja.es/servlet/articulo?codigo=2542432. Revisado diciembre 2008.

Anexos

Anexo A

Código para el manejo del periférico de colector abierto

A continuación se da una propuesta de aplicación de diseño en assembler para el periférico de colector abierto.

Aplicación en assembler

```
; Salida De Colector Abierto

Constant SALIDA, 00      ; DECLARO EL PUERTO DE SALIDA
Constant ENTRADA, 01     ; DECLARO EL PUERTO DE ENTRADA
INICIO:     INPUT     s0,   ENTRADA
            SUB       s0,   00
            JUMP      Z,    APAGADO   ; Envia a la rutina APAGADO
            SUB       s0,   01
            JUMP      Z,    SECUEN1   ; Primera secuencia
            ADD       s0,   01
            SUB       s0,   02
            JUMP      Z,    SECUEN2   ; Segunda secuencia
            JUMP      NZ,   SECUEN3   ; Tercera secuencia

APAGADO:    LOAD      s1,   00        ; Los leds están apagados
            OUTPUT    s1,   SALIDA
            JUMP      INICIO

SECUEN1:    LOAD      s1,   01        ; Los leds realizan un movimiento semejante a las
luces del auto fantastico
            OUTPUT    s1,   SALIDA
```

```
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   02
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   04
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   08
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   08
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   04
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   02
OUTPUT      s1,   SALIDA
CALL        delay_x0s
LOAD        s1,   00
OUTPUT      s1,   SALIDA
CALL        delay_x0s
```

```
                LOAD        s1,   01
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   00
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                JUMP        INICIO

SECUEN2:        LOAD        s1,   0F        ; Los leds prenden y apagan al mismo tiempo
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                JUMP        INICIO

SECUEN3:        LOAD        s1,   01        ; Los leds prenden de forma aleatoria
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   00
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   04
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   00
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   02
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   00
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   08
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   00
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                JUMP        INICIO

delay_x0s:      LOAD    s3,   19            ; Demora externa
```

```
        wait_x0s:       CALL    delay_xs
                        SUB     s3,     01
                        JUMP    NZ,     wait_x0s
RETURN

delay_xs:       LOAD    s2,     05              ; Demora intermedia
        wait_xs: CALL   delay_x1s
                        SUB     s2,     01
                        JUMP    NZ,     wait_xs
RETURN

delay_x1s:      LOAD    s4,     0A              ; Demora interna
        wait_x1s:SUB    s4,     01
                        JUMP    NZ,     wait_x1s
RETURN
```

Anexo B

Código para el manejo del periférico para el manejo de servomotores

A continuación se da una propuesta de módulo en VHDL y aplicación en assembler para el manejo de servomotores.

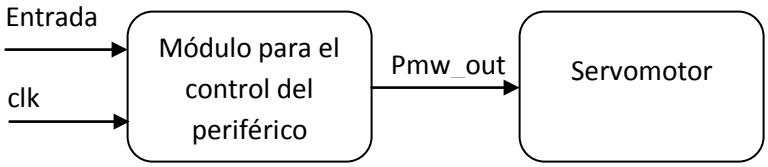

Figura 48. Diagrama de bloques del módulo manejo de servomotor.

Módulo en VHDL

```
Library IEEE;
Use IEEE.STD_LOGIC_1164.ALL;
Use IEEE.STD_LOGIC_ARITH.ALL;
Use IEEE.STD_LOGIC_UNSIGNED.ALL;
Entity modulo is

PORT (clk        : IN std_logic;
      Pulso      : buffer std_logic;
      Entrada    : in std_logic_vector (1 downto 0);
      pwm_out: out std_logic       );
end modulo;

architecture Behavioral of modulo is
```

```
signal count : std_logic_vector (7 downto 0)  := "00000000";
signal alto : std_logic_vector (4 downto 0 ) := "00000";

begin

divisor:
PROCESS (clk)

VARIABLE  cuenta    : INTEGER := 0;
CONSTANT modulo  : INTEGER := 210; --- Se garantiza el periodo de la señal

BEGIN
IF (clk'EVENT AND clk='1') THEN  -- Contador para el periodo
IF cuenta = modulo THEN
                cuenta := 0;
                pulso <= '1';
        ELSE
                cuenta := cuenta + 1;
                pulso <= '0';
        END IF;
END IF;
END PROCESS divisor;

pwm:
process (clk)       -- Se establece la señal del PWM
begin
if pulso ='1' then
                count <= "00000000";
        elsif clk'event and clk = '1' then
                count <= count + '1';
if count < alto then    -- Se establece el tiempo de alto
                        pwm_out <= '1';
                else
pwm_out <= '0';      -- Se establece el tiempo de bajo
                end if;
        end if;
end process pwm;


alto <=  "00000" when entrada  = "00"  else ---APAGADO EL SERVO
```

```
"01000" when entrada = "01" else ---POSICION DEL SERVO A -90 GRADOS TIEMPO = 8 ms
"10000" when entrada = "10" else ---POSICION DEL SERVO A 0 GRADOS TIEMPO = 2.0 ms
"11001" when entrada = "11"; ---POSICION DEL SERVO A 90 GRADOS TIEMPO   = 1.5 ms


----ESTOS TIEMPOS PARA UN RELOJ DE 10 KHz----

end Behavioral;
```

Aplicación en assembler

```
; Servomotor
Constant SALIDA, 00      ;DECLARO EL PUERTO DE SALIDA
constant ENTRADA, 01     ;DECLARO EL PUERTO DE ENTRADA

INICIO:     INPUT   s0,   ENTRADA   ; Se guarda la entrada en el registro S0
            SUB     s0,   00
            JUMP    Z,    STOP      ; Se envía a la rutina STOP
            SUB     s0,   01
            JUMP    Z,    MOV1      ; Movimiento 1
            ADD     s0,   01
            SUB     s0,   02
            JUMP    Z,    MOV2      ; Movimiento 2
            JUMP    NZ,   stop

STOP:       LOAD          s1,   00        ; En esta rutina el motor esta apagado
            OUTPUT        s1,   SALIDA
            JUMP          INICIO

MOV1:       LOAD          s1,   01        ; En esta rutina el motor realiza uno movimiento
desde 0 a 90 grados
            OUTPUT        s1,   SALIDA
            CALL          delay_x0s
            CALL          delay_x0s
            LOAD          s1,   03
            OUTPUT        s1,   SALIDA
            CALL          delay_x0s
            CALL          delay_x0s
```

```
                JUMP        INICIO

MOV2:           LOAD        s1,   02        ; En esta rutina el motor realiza un movimiento entre
0 y 180 grados
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                LOAD        s1,   03
                OUTPUT      s1,   SALIDA
                CALL        delay_x0s
                JUMP        INICIO

delay_x0s:    LOAD      s3, 19          ; Demora externa
    wait_x0s:   CALL    delay_xs
          SUB   s3, 01
                JUMP    NZ, wait_x0s
RETURN

delay_xs:     LOAD      s1, 05          ; Demora intermedia
    wait_xs:    CALL    delay_x1s
                SUB     s1, 01
                JUMP    NZ, wait_xs
RETURN

delay_x1s:    LOAD      s4, 0A          ; Demora interna
    wait_x1s:   SUB     s4, 01
                JUMP NZ, wait_x1s
RETURN
```

Anexo C

Código para el manejo del periférico conversor analógico/digital

A continuación se da una propuesta de módulo en VHDL y aplicación en assembler para el manejo del conversor analógico/digital.

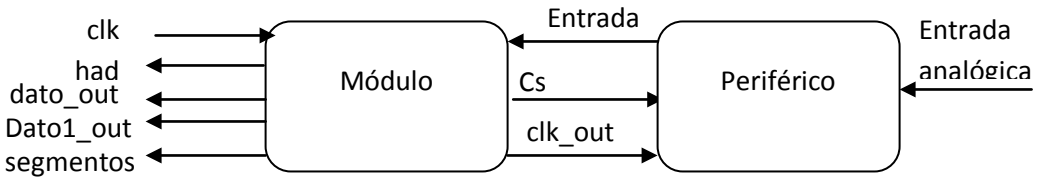

Figura 49. Diagrama de bloques del módulo conversor analógico/digital.

Módulo en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity PRUEBA9 is
PORT( clk            : IN std_logic; -- Entrada del reloj
      entrada        : in std_logic; -- Entrada analoga
      cs, clk_out    : out std_logic; -- Salida de habilitador y reloj para el periferico
```

```
        hab             : out std_logic; -- Habilitador para el display siete segmentos
                dato_out        : out std_logic_vector (6 downto 0); -- Vector para encender los leds de
la tarjeta
dato1_out       : out std_logic_vector (6 downto 0); -- Datos obtenidos de la conversión que van para el
procesador
segmentos       : out std_logic_vector (6 downto 0)); -- Datos para visualizar en el siete segmentos
end PRUEBA9;

architecture Behavioral of PRUEBA9 is

signal dato: std_logic_vector (15 downto 0) := (others => '0');
signal dato1: std_logic_vector (6 downto 0) := (others => '0');
signal segmentos1: std_logic_vector (6 downto 0) := (others => '0');
signal segmentos2: std_logic_vector (6 downto 0) := (others => '0');

begin
PROCESS (clk)
        VARIABLE cuenta : INTEGER:= 0;
        CONSTANT modulo : INTEGER:= 16;--- ESTA GARANTIZANDO EL PERIODO
BEGIN
IF (clk'EVENT AND clk='0') THEN -- CONTADOR PARA EL PERIODO
                IF cuenta = modulo THEN
                        cuenta := 0;
                        cs <= '1';
dato_out <= not(dato (10 downto 4));
dato1_out <= dato (10 downto 4);
dato1 <= dato (10 downto 4);
                ELSE
                        cs <= '0';
                        cuenta := cuenta + 1;
dato <= dato(14 downto 0) & entrada;
                END IF;
        END IF;
END PROCESS;

clk_out <= clk;

---Módulo siete segmentos:
segmentos1 <="1111110" when (dato1 < x"02") or (dato1 > x"24" and dato1 < x"29") or (dato1 > x"4B"
                and dato1 < x"4F")or (dato1 > x"71" and dato1 < x"76")else               --0
```

```
                "0110000" when (dato1 > x"01" and dato1 < x"06") or (dato1 > x"28" and dato1 < x"2D") or
                  (dato1 > x"4E" and dato1 < x"53")or (dato1 > x"75" and dato1 < x"7A")else   --1
                "1101101" when (dato1 > x"05" and dato1 < x"0A") or (dato1 > x"2C" and dato1 < x"31") or
                  (dato1 > x"52" and dato1 < x"57")or (dato1 > x"79" and dato1 < x"7E")else   --2
                "1111001" when (dato1 > x"09" and dato1 < x"0E") or (dato1 > x"30" and dato1 < x"34") or
                  (dato1 > x"56" and dato1 < x"5B")or (dato1 > x"7D" and dato1 < x"80")else   --3
                "0110011" when (dato1 > x"0D" and dato1 < x"12") or (dato1 > x"33" and dato1 < x"38") or
                  (dato1 > x"5A" and dato1 < x"5F")else   --4
                "1011011" when (dato1 > x"11" and dato1 < x"16") or (dato1 > x"37" and dato1 < x"3C") or
                  (dato1 > x"5E" and dato1 < x"63")else   --5
                "1011111" when (dato1 > x"15" and dato1 < x"1A") or (dato1 > x"3B" and dato1 < x"40") or
                  (dato1 > x"62" and dato1 < x"66")else   --6
                "1110000" when (dato1 > x"19" and dato1 < x"1D") or (dato1 > x"3F" and dato1 < x"44")
                  or (dato1 > x"65" and dato1 < x"6A")else   --7
                "1111111" when (dato1 > x"1C" and dato1 < x"21") or (dato1 > x"43" and dato1 < x"48") or
                  (dato1 > x"69" and dato1 < x"6E")else   --8
                "1110011" when (dato1 > x"20" and dato1 < x"25") or (dato1 > x"47" and dato1 < x"4C") or
                  (dato1 > x"6D" and dato1 < x"72")else   --9
                "1111111";
segmentos2  <= "1111110" when dato1 < x"25" else
                "0110000" when dato1 > x"24" and dato1 < x"4C" else
                "1101101" when dato1 > x"4B" and dato1 < x"72" else
                "1111001";

segmentos <= segmentos1 when clk = '0' else segmentos2;

hab <= '0' when clk = '0' else '1';

end Behavioral
```

Aplicación en assembler

```
;Conversor A/D
Constant SALIDA,        00      ; DECLARO EL PUERTO DE SALIDA
constant ENTRADA,       01      ;DECLARO EL PUERTO DE ENTRADA

INICIO: INPUT  s0,      ENTRADA
SUB     s0,     4B
```

```
        JUMP    Z,      BAJO    ; El rango de entrada es menor de 2 [V]
        ADD     s0,     4B
        SUB     s0,     75
        JUMP    Z,      ALTO    ; El rango de entrada es mayor de 3 [V]
        JUMP    NZ,     INRAN   ;El rango de entrada se encuentra entre 2 [V] y 3 [V]

BAJO:   LOAD            s1,     01      ;Cuando la entrada esta es un nivel bajo entonces se enciende
un led
        OUTPUT          s1,     SALIDA
        JUMP            INICIO

ALTO:   LOAD            s1,     02  ; Cuando la entrada esta en un nivel alto se enciende otro led
        OUTPUT          s1,     SALIDA
        JUMP            INICIO

INRAN:  LOAD            s1,     FF  ; Cuando la entrada esta dentro de los niveles no se enciende los led
        OUTPUT          s1,     SALIDA
        JUMP            INICIO
```

Anexo D

Código para el manejo del periférico conversor digital/analógico

A continuación se da una propuesta de módulo en VHDL y aplicación en assembler para el manejo del conversor digital/analógico.

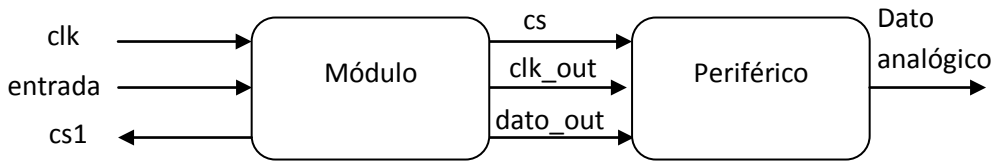

Figura 50. Diagrama de bloques del módulo conversor digital/analógico.

Módulo en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity modulo is
PORT(       clk : IN std_logic;  -- Entrada del reloj de la tarjeta
            entrada: in std_logic_vector (7 downto 0);      -- Entrada digital de datos a convertir
```

```
        cs, clk_out, cs1 : out std_logic;          -- Salida de: habilitar para el periférico, reloj para el periférico y habilitar para indicarle al procesador que los datos ya están convertidos y se pueden enviar los siguientes
        dato_out: out std_logic        ); -- Dato digital enviado al periférico para ser convertido
end modulo;

architecture Behavioral of modulo is

signal dato1: std_logic_vector (15 downto 0) := (others => '0');

begin

PROCESS (clk)
    VARIABLE    cuenta    : INTEGER    :=    0;
    CONSTANT    compara   : INTEGER    :=    16;    --- Esta garantizando periodo
BEGIN
    IF (clk'EVENT AND clk='1') THEN              -- Contador para el periodo
        IF cuenta = compara THEN
            cuenta := 0;
            cs <= '1';
            cs1 <= '1';
            dato1 <= x"0" & entrada & x"F";
        ELSE
            cs <= '0';
            cs1 <= '0';
            cuenta := cuenta + 1;
            dato_out <= dato1(15);
            dato1 <= dato1 (14 downto 0) & '0';
        END IF;
    END IF;
END PROCESS;

clk_out <= clk;

end Behavioral;
```

Aplicación en assembler

Para poner en funcionamiento estas aplicaciones es necesario copiar el código respectivo para cada señal, ya que no es posible debido a la capacidad de la memoria del procesador.

```
;Conversor D/A
Constant        SALIDA,         00    ; DECLARO EL PUERTO DE SALIDA
Constant        ENTRADA,        01    ;DECLARO EL PUERTO DE ENTRADA

; Señal cuadrada
inicio:  INPUT   s0,     ENTRADA              ; Guardo la entrada en el registro S0
         SUB     s0,     00
         JUMP    NZ,     INICIO
         LOAD    s5,     7F                   ; Se envia para hacer el nivel alto de la señal
         OUTPUT          s5,     SALIDA
inicio1:         INPUT   s0,     ENTRADA
                 SUB      s0,    00
                 JUM      NZ,    INICIO1
                 LOAD     s5,    00           ; Se envia para hacer el nivel bajo de la señal
                 OUTPUT s5,      SALIDA
                 JUMP     inicio

; Señal rampa
INICIO:          LOAD            s1,     00
INICIO1:         INPUT           s0,     ENTRADA
                 SUB             s0,     00
                 JUMP            NZ,     INICIO1 ; Se verifica que la señal halla llegado su máximo
nivel de alto analogico
                 OUTPUT          s1,     SALIDA
                 ADD             s1,     01
                 SUB             s1,     7F
                 JUMP            Z,      INICIO
                 ADD             s1,     7F
                 JUMP            NZ,     INICIO1
```

```
; señal senoidal
INICIO1:    LOAD        s1,     39      ; Se envian los datos tabulados para generar la señal
senoidal
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     3D
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     41
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     45
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     48
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     4C
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     4F
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     52
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     55
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     57
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     5A
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     5C
            OUTPUT      s1,     SALIDA
            CALL        INICIO
            LOAD        s1,     5D
```

```
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5E
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5F
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    60
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    60
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5F
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5F
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5E
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5C
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    5A
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    58
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    56
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    53
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    50
OUTPUT    s1,    SALIDA
```

```
CALL    INICIO
LOAD    s1,  4D
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  4A
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  46
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  42
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  3F
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  3B
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  37
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  33
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  2F
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  2E
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  28
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  25
OUTPUT  s1,  SALIDA
CALL    INICIO
LOAD    s1,  22
OUTPUT  s1,  SALIDA
CALL    INICIO
```

```
LOAD        s1,     1F
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     1C
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     1A
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     18
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     16
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     15
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     14
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     13
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     13
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     13
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     13
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     14
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     16
OUTPUT      s1,     SALIDA
CALL        INICIO
LOAD        s1,     17
```

```
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    19
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    1B
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    1E
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    21
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    24
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    27
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    2B
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    2E
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    32
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    36
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    3A
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    3A
OUTPUT    s1,    SALIDA
CALL      INICIO
LOAD      s1,    3A
OUTPUT    s1,    SALIDA
```

```
                JUMP        INICIO1

INICIO:         INPUT   s0, ENTRADA
                SUB     s0,  00
                JUMP    NZ, INICIO
                RETURN
```

Anexo E

Código para el manejo del periférico del puente H

A continuación se da una propuesta de módulo en VHDL y aplicación en assembler para el manejo del puente H.

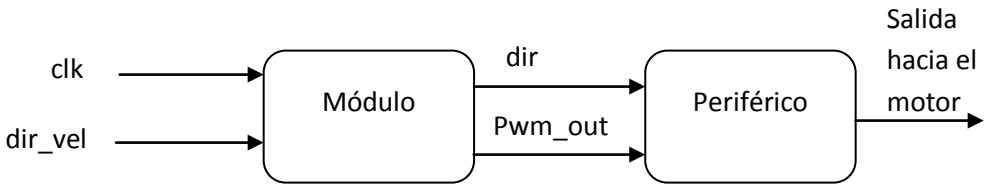

Figura 51. Diagrama de bloques del módulo puente H.

Módulo en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY modulo IS
PORT(
        clk         : IN std_logic;       -- Reloj que proviene de la tarjeta
        pulso       : buffer std_logic;   -- Señal que es de salida pero puede ser leída
        dir_vel     : in std_logic_vector(2 downto 0);      -- Vector proveniente del procesador y cual tiene la
información de la velocidad y de la dirección que debe tomar el motor
        pwm_out, direc      : out std_logic);     -- PWM que controlara el motor
END modulo;

ARCHITECTURE behavorial OF modulo IS
signal count: std_logic_vector (2 downto 0) :="000";
```

```
signal pwm_in: std_logic_vector (2 downto 0) :="000";

BEGIN
divisor:
PROCESS (clk)
VARIABLE cuenta : INTEGER;
CONSTANT modulo : INTEGER      :=          8;

BEGIN                          ---para el period del PWM
        IF (clk'EVENT AND clk='1') THEN
                IF cuenta = modulo THEN
                        cuenta :=  0;
                        pulso      <= '1';
                ELSE
                        cuenta := cuenta + 1;
                        pulso      <= '0';
                END IF;
        END IF;
END PROCESS divisor;

pwm:
process (clk)                  ---Para el ciclo util del pwm (tiempo en nivel alto)
begin
        if pulso = '1' then
                count <= "000";
        elsif clk'event and clk = '1' then
                count <= count + '1';
                if count < pwm_in then
                        pwm_out <= '1';
                else
                        pwm_out <= '0';
                end if;
        end if;
end process pwm;

pwm_in <=       "000" when dir_vel             = "000" else
                "100" when dir_vel(2) = '0'   else
                "111" when dir_vel(2) = '1';

direc <= dir_vel(0);
```

```
END behavorial
```

Aplicación en assembler

```
; Puente h
Constant SALIDA, 00      ; Declaro el puerto de salida
constant ENTRADA, 01     ;Declaro el puerto de entrada

INICIO: INPUT  s0        ENTRADA        ; Guardo le entrada en el registro S0
        SUB    s0,       00
        JUMP   Z,        STOP           ; Si no hay entrada se envía cero al módulo
        SUB    s0        01
        JUMP   Z,        MOV1           ; Velocidad 1
        ADD    s0,       01
        SUB    s0,       02
        JUMP   Z,        MOV2           ; Velocidad 2
        JUMP   NZ,       stop

STOP:  LOAD              s1,       00          ; Rutina en la que se le envía al módulo cero
        OUTPUT           s1,       SALIDA
        JUMP             INICIO

MOV2: LOAD               s1,       05          ; Se establece una velocidad y giro
        OUTPUT           s1,       SALIDA
        CAL              delay_x0s
        CALL             delay_x0s
        LOAD             s1,       00          ; Se apaga el motor para hacer el cambio de giro
        OUTPUT           s1,       SALIDA
        CALL             delay_x0s
        CALL             delay_x0s
        LOAD             s1,       04          ; Se establece la misma velocidad pero se cambia  de giro
        OUTPUT           s1,       SALIDA
        CALL             delay_x0s
        CALL             delay_x0s
        LOAD             s1,       00          ;Se apaga el motor
        OUTPUT           s1,       SALIDA
        CALL             delay_x0s
        CALL             delay_x0s
```

```
        JUMP        INICIO

MOV1:   LOAD        s1,     01
        OUTPUT      s1,     SALIDA
        CALL        delay_x0s
        CALL        delay_x0s
        LOAD        s1,     00
        OUTPUT      s1,     SALIDA
        CALL        delay_x0s
        CALL        delay_x0s
        LOAD        s1,     02
        OUTPUT      s1,     SALIDA
        CALL        delay_x0s
        CALL        delay_x0s
        LOAD        s1,     00
        OUTPUT      s1,     SALIDA
        CALL        delay_x0s
        CALL        delay_x0s
        JUMP        INICIO

delay_x0s:  LOAD        s3,     19      ; Demora externa
wait_x0s:   CAL         delay_xs
            SUB         s3,     01
            JUMP        NZ,     wait_x0s
        RETURN

delay_xs:   LOAD        s2,     05      ; Demora intermedia
wait_xs:    CALL        delay_x1s
            SUB         s2,     01
            JUMP        NZ,     wait_xs
        RETURN

delay_x1s:  LOAD        s4,     0A      ; Demora interna
wait_x1s:   SUB         s4,     01
            JUMP        NZ,     wait_x1s
        RETURN
```

Anexo F

Código para el manejo del periférico del amplificador de audio

A continuación se da una propuesta de módulo en VHDL y aplicación en assembler para el manejo del amplificador de audio.

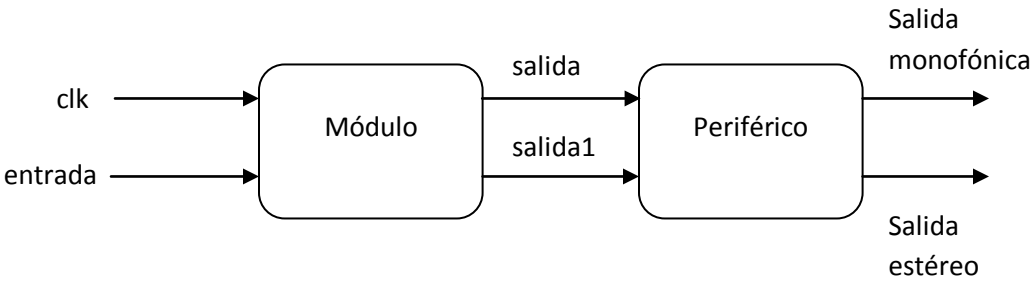

Figura 52. Diagrama de bloques del módulo amplificador de audio.

Módulo en VHDL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity modulo is

port (    clk              :          in  std_logic;
          entrada          :           in std_logic_vector(3 downto 0);--señal que viene del procesador
          salida, salidal  :          out std_logic         );--dos salidas mono y stereo
end modulo;
```

```
architecture Behavioral of modulo is

SIGNAL  contador        :  std_logic_vector(15 downto 0);
SIGNAL  contador2       :  std_logic_vector(15 downto 0);
SIGNAL  alto            :  std_logic_vector(4 downto 0);

begin

process (clk)
        begin                           -- divisor de frecuencia para las notas musicales
                if(rising_edge(clk)) then
                        if(contador > alto) then
                                contador <= (others =>'0');
                                contador2 <= contador2+1;
                        else
                                contador <= contador+1;
                        end if;
                end if;
end process;

process (entrada)

begin
        case entrada is         ----dependiendo la entrada resetea el contador para cada nota
        WHEN            "0001"    => alto <= "10010";
        WHEN             "0010"   => alto <= "10000";
        WHEN             "0011"   => alto <= "01110";
        WHEN             "0100"   => alto <= "01101";
        WHEN             "0101"   => alto <= "01100";
        WHEN             "0110"   => alto <= "01010";
        WHEN             "0111"   => alto <= "01001";
        WHEN             "0000"   => alto <= "00000";
        WHEN OTHERS                => alto <= "00000";
        END CASE;
end process;

salida  <= '0' when alto = "00000" else contador2(0);
salida1 <= '0' when alto = "00000" else contador2(0);
```

```
end Behavioral;
```

Aplicación en assembler

```
;Amplificador de audio, canción de la alegría

Constant salida, 00      ; DECLARO EL PUERTO DE SALIDA
constant entrada, 01     ; DECLARO EL PUERTO DE ENTRADA


INICIO: input    s0,      entrada  ; Se verifica que se halla seleccionado escuchar la canción
        sub      s0,      01
        jump     z,       INICIO1  ; Si se puso un uno en la entrada se reproduce la cancion
        jump     INICIO

INICIO1: load    s1,      03       ; Se empiezan a enviar la secuencia de las notas para la canción
        output   s1,      salida
        call     delay_x0s
        load     s1,      04
        output   s1,      salida
        call     delay_x0s
        load     s1,      05
        output   s1,      salida
        call     delay_x0s
        load     s1,      04
        output   s1,      salida
        call     delay_x0s
        load     s1,      03
        output   s1,      salida
        call     delay_x0s
        load     s1,      02
        output   s1,      salida
        call     delay_x0s
        load     s1,      01
        output   s1,      salida
        call     delay_x0s
        load     s1,      02
```

```
output  s1,  salida
call    delay_x0s
load    s1,  03
output  s1,  salida
call    delay_x0s
 load   s1,  02
output  s1,  salida
call    delay_x0s
 load   s1,  00
output  s1,  salida
call    delay_x0s
 load   s1,  03
output  s1,  salida
call    delay_x0s
load    s1,  04
output  s1,  salida
call    delay_x0s
load    s1,  05
output  s1,  salida
call    delay_x0s
load    s1,  04
output  s1,  salida
call    delay_x0s
load    s1,  03
output  s1,  salida
call    delay_x0s
load    s1,  02
output  s1,  salida
call     delay_x0s
load    s1,  01
output  s1,  salida
call    delay_x0s
load    s1,  02
output  s1,  salida
call    delay_x0s
load    s1,  03
output  s1,  salida
call    delay_x0s
load    s1,  02
output  s1,  salida
```

```
        call    delay_x0s
        load    s1,     01
        output  s1,     salida
        call    delay_x0s
        load    s1,     00
        output  s1,     salida
        call    delay_x0s
        jump    INICIO

delay_x0s:  LOAD    s3,     06      ; Demora externa
wait_x0s:   CALL    delay_xs
            SUB     s3,     01
            JUMP    NZ,     wait_x0s
RETURN

delay_xs:   LOAD    s1,     05      ; Demora intermedia
wait_xs:    CALL    delay_x1s
            SUB     s1,     01
            JUMP    NZ,     wait_xs
RETURN

delay_x1s:  LOAD    s4,     0A      ; Demora interna
wait_x1s:   SUB     s4,     01
            JUMP    NZ,     wait_x1s
RETURN
```

Anexo G

Código para el manejo del periférico del puerto RS232

A continuación se da una propuesta de módulo en VHDL y aplicación en assembler para el manejo del puerto RS232.

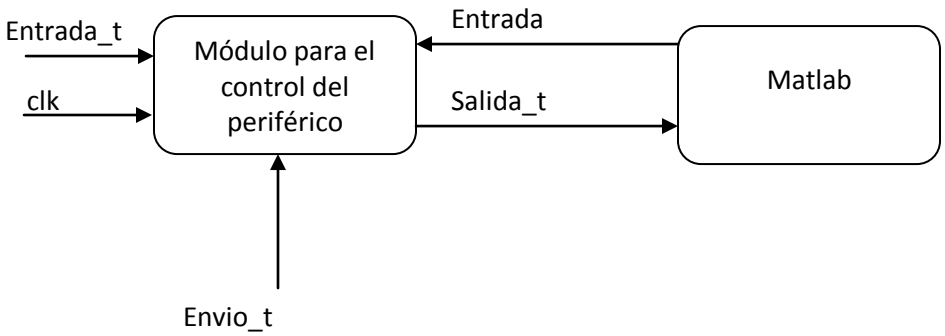

Figura 53. Diagrama de bloques del módulo RS232.

Módulo en VHDL

```
--MODULO RS232
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity modulo is

PORT(clk, envio_t : in std_logic;
                entrada_t : in std_logic_vector (7 downto 0);
```

```
                blanco_t :  out std_logic;
                salida1_t : out std_logic;
                entrada : in std_logic;------recepción
                blanco :  out std_logic;
                mostrar : out std_logic_vector(7 downto 0) );
end modulo;


architecture Behavioral of modulo is

--transmisión----------------------------------------------------
----------------------------------------------------------------
TYPE estado_t IS (inicio_t, contando_t);
signal presente_t : estado_t:= inicio_t;
signal contador_t : std_logic_vector(7 downto 0) := (others => '0');
signal enable_t  : std_logic := '0';
signal enable1_t : std_logic;
signal entrada1_t : std_logic_vector(10 downto 0):= "00000000000";
--fin transmisión-----------------------------------------------
----------------------------------------------------------------

--recepción-----------------------------------------------------
----------------------------------------------------------------
TYPE estado IS (inicio, contando);
signal presente : estado:= inicio;
signal contador : std_logic_vector(7 downto 0) := (others => '0');
signal enable  : std_logic := '0';
signal enable1 : std_logic;
signal dato : std_logic_vector(99 downto 0):= (others => '0');
--fin recepción-------------------------------------------------
----------------------------------------------------------------

--divisor de frecuencia-----------------------------------------
----------------------------------------------------------------
signal contador_clk : std_logic_vector(7 downto 0):= (others => '0');
signal clk2 : std_logic;
signal contador2 : std_logic_vector(40 downto 0):= (others => '0');
--fin divisor de frecuencia-------------------------------------
----------------------------------------------------------------
```

```
begin

--transmisión-------------------------------------------------------
-------------------------------------------------------------------
maquina_t:
process(clk)
begin
        if(rising_edge(clk2)) then

                case presente_t is

                when inicio_t =>

                        if (envio_t = '1') then
                                presente_t <= contando_t;
                                enable1_t <= '0';
                        else
                                presente_t <= inicio_t;
                                entrada1_t <= "010" & entrada_t;
                                                ------"01010001010";
                                                                ----12345678
                        end if;

                when contando_t =>
                        if (enable_t = '1') then
                                presente_t <= inicio_t;
                                enable1_t <= '1';
                        else
                                presente_t <= contando_t;
                                salida1_t <= entrada1_t (10);
                                entrada1_t <= entrada1_t(9 downto 0) & '0';

                        end if;

                end case;
        end if;
end process maquina_t;

salida_t:
process(presente_t)
```

```
begin
        case presente_t is

        when inicio_t =>

                blanco_t <= '0';

        when contando_t =>

                blanco_t <= '1';

        end case;
end process salida_t;

recibir_dato_t:
process(clk2,enable1_t)
begin
if (enable1_t = '0') then
if (rising_edge(clk2)) then
 if (contador_t < 12)          and (enable1_t = '0') then
        contador_t <= contador_t+1;

 else
        contador_t <= "00000000";

  end if;
 end if;
end if;
end process recibir_dato_t;
enable_t <= '1' when contador_t = 11   else '0';
--fin transmisión-------------------------------------
-----------------------------------------------------


--recepción-----------------------------------------
-----------------------------------------------------
maquina:
process(clk)
begin
        if(rising_edge(clk)) then
```

```
                case presente is

                when inicio =>

                        if (entrada = '0') then
                                presente <= contando;
                                enable1 <= '0';
                        else
                                presente <= inicio;
                        end if;

                when contando =>
                        if (enable = '1') then
                                presente <= inicio;
                                enable1 <= '1';
                        else
                                presente <= contando;
                                dato <= dato(98 downto 0) & entrada;
                        end if;

                end case;
        end if;
end process maquina;

salida:
process(presente)
begin
        case presente is

        when inicio =>

                blanco <= '0';
                mostrar(0) <= not dato(15);
                mostrar(1) <= not dato(25);
                mostrar(2) <= not dato(35);
                mostrar[3] <= not dato(45);
                mostrar(4) <= not dato(55);
                mostrar(5) <= not dato(65);
                mostrar(6) <= not dato(75);
```

```
                mostrar(7) <= not dato(85);

        when contando =>

                blanco <= '1';


        end case;
end process salida;

recibir_dato:
process(clk)
begin
if (enable1 = '0') then
if (rising_edge(clk)) then
 if (contador < 102)          and (enable1 = '0') then
        contador <= contador+1;

 else
        contador <= "00000000";

  end if;
 end if;
end if;
end process recibir_dato;

enable <= '1' when contador = 100   else '0';
- - fin recepción-----------------------------------
-------------------------------------------------

--  divisor de frecuencia----------------------------
-------------------------------------------------
process(clk)
begin
        if(rising_edge(clk)) then
                if(contador_clk = 10)then
                        contador_clk <= (others=> '0');
                        contador2 <= contador2+1;
                else
                        contador_clk <= contador_clk+1;
```

```
                end if;
            end if;
end process;
clk2 <= '1' when contador_clk = 9 else '0';

end Behavioral;
```

Aplicación en assembler

```
 ;Comunicacion RS232 con Matlab

Constant salida, 01      ; DECLARO EL PUERTO DE SALIDA
constant entrada, 03     ;DECLARO EL PUERTO DE ENTRADA

INICIO:   input s0, entrada          ; Guardo la entrada en el registro S0
          output s0, salida          ; Pongo lo guardado en el registro S0 en la salida
          jump inicio
```

Código empleado en Matlab

```
clc
clear

global count;

disp('PRUEBA DE COMUNICACION SERIAL');
disp('ACTIVACION Y CONFIGURACION DEL PUERTO');

s = serial('com8')          % avilitando el puerto serial com8
s.BaudRate=9000;            % configuracion del Rata de Baudios
s.OutputBufferSize=8;       % tamaño del buffer
s.InputBufferSize=1;        % tamaño del buffer
fopen(s);                   % abriendo el puerto

disp('LISTO PARA RECIBIR')
```

```
fprintf('\n\n');
for (j=0:10)                % Numero de comunicaciones entre el CPLD y Mattlab
a=0;

set(s,'Timeout',60)         % Tiempo en que Matlab espera el dato
while(a == 0)               % Si el dato de entrada es cero no se tiene en cuenta
   a = (fread(s));          % Lee el puerto y lo guarda en la variable a
end
disp('DATO RECIBIDO ');
disp(a);
disp('DATO ENVIADO ');
b = round(char(a)/2);       % Cambia de formato la variable a, divide en 2 y reondea al numero entero
                            siguiente
disp(b);

fprintf(s,'%c',b);          % Escribe en el puerto el dato b

fprintf('\n\n\n');
end


fclose(s);                  % Cierra el puerto
```