DISEÑO E IMPLEMENTACIÓN DE LA TARJETA PARA CONTROL DE ROBOT PUMA MA2000

Nicolás Fernando Beltrán Garavito
David Miguel Medina Rojas

UNIVERSIDAD INDUSTRIAL DE SANTANDER

ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES

Bucaramanga – 2011

DISEÑO E IMPLEMENTACIÓN DE LA TARJETA PARA CONTROL DE ROBOT PUMA MA2000

Nicolás Fernando Beltrán Garavito
David Miguel Medina Rojas

Trabajo para optar por el título de Ingeniero Electrónico

Director
MSc. Jorge Hernando Ramón Suarés

Codirector
Dr. Rodolfo Villamizar

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍAS ELÉCTRICA, ELECTRÓNICA Y TELECOMUNICACIONES
Bucaramanga, Febrero de 2011

.

A Dios, porque en él se fundamenta todo.

A mi madre, por todo el amor, la comprensión, el apoyo y el ejemplo que ha formado lo mejor de mí.

A mi hermana y en general a mi Familia, por su desquisiado e incondicional cariño y compañia.

A todos aquellos que poseen una parte de mi corazón, porque los momentos vividos juntos son los más valiosos del mundo.

A todos mis profesores y compañeros, por sus enseñanzas, sus consejos, los conocimientos y el tiempo compartidos.

Al Grupo ERA por ser el centro donde he logrado aplicar los conocimientos adquiridos a lo largo de la carrera.

Al Programa PAMRA por darme la oportunidad de transmitir todo el conocimiento y la experiencia a las siguientes generaciones.

A la Universidad Industrial de Santander que a esta altura es más que mi segundo hogar.

Nicolás Fernando Beltrán Garavito

.

A mi mamá, que siempre ha estado a mi lado
brindandome el cariño y el apoyo necesario
para lograr mis metas.

A mi abuela, gracias a sus consejos,
que siempre me han hecho seguir adelante.

A mi hermana, que siempre ha mostrado
su inmenso cariño y apoyo.

A las personas que se encuentran
en mi corazón y que siempre me han apoyado.

A los miembros de la comunidad universitaria
que dieron los consejos necesarios para
encaminarme en el camino del éxito.

Al Grupo ERA que me brindo los elementos
necesarios para desarrollar mis capacidades.

Este es el fruto del amor y la comprensión de mi familia,
gracias por formar la persona que soy.

David Miguel Medina Rojas

Índice general

Índice de Figuras

Índice de Tablas

Resumen

Titulo:
Diseño e Implementación de la Tarjeta para Control de Robot PUMA MA2000[1] [2]

Autores[3]:
Nicolás Fernando Beltrán Garavito
David Miguel Medina Rojas

Palabras Clave: Sistema Empotrado, FPGA, RTOS, CPU, Robot tipo PUMA, PID, Aritmética Distribuida.

Contenido:

En la actualidad el diseño de sistemas empotrados ha tomado un ritmo acelerado que implica que cada decisión sea critica en la salida de un producto al mercado, esto se ha producido debido a la demanda de este tipo de sistemas en la industria, educación, así como para el uso personal. Uno de las aplicaciones que ha mostrado un crecimiento acelerado en los últimos tiempos, son las plataformas para control de sistemas físicos.

En este documento se muestra el diseño e implementación de un sistema empotrado prototipo para el control de un brazo robótico tipo PUMA de seis grados de libertad, referencia MA 2000 de la empresa TecQuiment LTD.

En el capitulo 1 se muestra la iniciativa que hizo posible este trabajo y los respectivos objetivos que se plantearon al inicio del proyecto. En el capitulo 2 se encuentra una recopilación de información valiosa a la hora de diseñar un sistema empotrado. En el capitulo 3 se muestra el diseño del prototipo y de cada uno de los componentes que hacen parte de este. En el capitulo 4 se encuentran consignadas las pruebas realizadas que permitieron la validación de cada uno de los módulos diseñados. Finalmente en el capitulo 5 se presentan las conclusiones del trabajo realizado y las recomendaciones para trabajos futuros.

Adicionalmente se incluyen anexos con información sobre RTOS disponibles hasta la fecha de realización del proyecto en el mercado, esquemáticos que ilustran los elementos hardware adicionales que permiten el funcionamiento del sistema inmerso dentro de la FPGA, así como los códigos utilizados más relevantes a la hora de realizar las pruebas.

[1] Proyecto de Investigación para optar por el titulo de Ingeniero Electrónico

[2] Director: Msc. Jorge Hernando Ramón Suarez, CoDirector: Dr. Rodolfo Villamizar

[3] Miembros de la Facultad de Ingenierías Físico Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones.

Abstract

Title:
Design and Implementation of the MA2000 PUMA Robot Control Card[4] [5] [6]

Authors[7]: Nicolás Fernando Beltrán Garavito
David Miguel Medina Rojas

Keywords: Embedded System, FPGA, RTOS, CPU, PUMA Robot, PID, Distributed Arithmetic.

Description:

Nowadays embedded systems have got an exponential growing that makes critical each desition in order to get a good release of a product into market. This growing is due the rising market demand of these kind of products in industry, education and in personal consume. Control cards for physical systems is one of the application that had most grown its market demand.

In this project the design and implementation of an embedded system to control a MA2000 PUMA robotic arm with six degrees of freedom, manufactured by TecQuipment LTD, is developed.

The initiative that made this work possible and the definition of the objetives of the whole project are shown at Chapter 1. The most relevant documental information in order to design an embedded system is collected in Chapter 2. Chapter 3 describes the design of the prototype and its main modules. In Chapter 4 it can be found the main test to validate each module and the total system. Finally the conclusions of the present proyect and the advices for future works are referenced in Chapter 5.

Aditionally, some annexes are added to give information like: Actual RTOS availability in the market at project development time, work schemes of the additional hardware that allows the embedded system to work in an FPGA architecture, and the main testing codes.

[4]Research Project to qualify for the Electronic Engineering degree

[5]Director: Msc. Jorge Hernando Ramón Suarez

[6]Co-director: Dr. Rodolfo Villamizar

[7]Belog to the Physical-Mecanical Engineering Faculty. School of Electrical, Electronics and Telecommunications Engineerings.

Capítulo 1

Preliminares

1.1. Planteamiento del Problema

La Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones adquirió en el año de 1995 un conjunto de tres equipos con control numérico por computadora (CNC) a la empresa TecQuipment Ltd., con los cuales se buscaba incentivar el aprendizaje de esta técnica presente en la industria para la automatización de procesos de maquinado. Sin embargo el usuario no tenía fácil acceso a las estrategias de control cinemático y dinámico de estos equipos y tampoco podía agregar funcionalidad al software adquirido con la CNC.

Los equipos anteriormente mencionados entraron en desuso debido al daño de hardware y software en el equipo de cómputo central y a la falta de soporte por parte del fabricante.

Debido a que el robot PUMA referencia MA2000 es el encargado de movilizar las piezas a ser maquinadas y a su aplicabilidad como herramienta académica se desea poner nuevamente en funcionamiento cumpliendo los siguientes requerimientos:

- Tiempo de cómputo para acciones sucesivas de control no mayor a 10 [ms].
- Controladores de código y hardware abierto.
- Funcionamiento de los controladores en paralelo.
- Capacidad para recepción de órdenes de control de forma remota.

1.2. Objetivos

1.2.1. Principal

Diseñar e implementar la tarjeta de control para un robot PUMA, referencia MA2000 de seis grados de libertad, con soporte para sistemas operativos de tiempo real y capacidad para la recepción de instrucciones de forma directa y remota.

1.2.2. Específicos

Para el cumplimiento del objetivo general del proyecto se establecieron las siguientes acciones:

- Establecer los requerimientos de hardware y software para la implementación de un Sistema Operativo de Tiempo Real en un sistema empotrado.
- Diseñar la tarjeta de control contemplando los requerimientos del sistema a manejar, los estándares de la interfaz en la tarjeta, así como el tratamiento y adecuación de las señales de salida a los actuadores.
- Validar la satisfacción de los requerimientos en la tarjeta implementada, mediante la ejecución de una serie de instrucciones, administradas por un sistema operativo de tiempo real.

1.3. Justificación

El presente trabajo de grado surge como respuesta a la necesidad de mejorar y adecuar la plataforma MA2000 disponible en los laboratorios de la escuela de ingenierías eléctrica, electrónica y de telecomunicaciones, la cual no se encuentra en funcionamiento en la actualidad, debido a daños en el hardware y obsolescencia de software, indispensable para su funcionamiento.

Con la implementación de una nueva tarjeta de control, se mejorarán los medios de recepción de instrucciones, así como la realimentación al usuario del comportamiento del sistema. Igualmente, el brindar capacidad de reprogramación de la dinámica y la cinemática por separado incrementa la versatilidad de este.

Asegurar en la tarjeta el soporte para uso de un sistema operativo de tiempo real permite la adecuación de la tarjeta al manejo de múltiples tareas en forma paralela de forma eficiente, optimizando el uso de los recursos, reduciendo la susceptibilidad a errores de la misma y asegurando la ejecución de tareas en los tiempos especificados.

Debido a la necesidad de ejecutar tareas en paralelo se requiere una tarjeta de control que permita ejecutar el número de operaciones necesarias en el tiempo pertinente cumpliendo con la sensación de ejecución en tiempo real.

La construcción de esta tarjeta esta fundamentada en la necesidad de tener una plataforma de software y hardware abiertos, ya que esta plataforma brindará oportunidades para la investigación y aplicación de conocimientos en las áreas de: sistemas operativos de tiempo real, técnicas de control, teleoperación de dispositivos y diseño de sistemas empotrados capaces de ejecutar la carga computacional necesaria para realizar las tareas de alto nivel antes mencionadas.

La ejecución de este proyecto indagará además en el conocimiento necesario para la generación de criterios de selección de una Unidad Central de Procesamiento.

Capítulo 2

Antecedentes

2.1. Metas del Diseño

Un proyecto que implique un sistema empotrado debe tener metas claras, para tener indicadores del trabajo realizado, así como un punto de referencia hacia donde se quiere llegar. A la hora de proponer las metas se deben contemplar los siguientes indicadores[2]:

1. Desempeño: En un sistema empotrado puede ser medido de varias formas,
   - Desempeño promedio contra el peor caso o el mejor caso.
   - Throughput[1] contra latencia.
   - Rendimiento pico contra rendimiento medio.
2. Potencia: El sistema debe tener en cuenta el consumo del dispositivo ya que generalmente estará alimentado con baterías así como la producción de calor del circuito.
3. Costos: Es posible medirlos por medio del costo de manufactura, costo de diseño, incluyendo todo el soporte necesario para mantener el equipo de diseño y el costo durante el tiempo de vida, que implica el mantenimiento y actualizaciones hardware y software.
4. Tiempo de Diseño: Parámetro importante, ya que el mercado objetivo puede verse alterado en el tiempo.
5. Fiabilidad: Depende del mercado objetivo que se seleccione.
6. Calidad: La calidad es importante pero puede causar dificultades a la hora de definirse y medirse. Puede estar relacionada con la fiabilidad en algunos mercados así como puede estar relacionada con la interacción del usuario, es relativa al mercado objetivo.

[1] Volumen de datos procesados.

2.2. Metodologías de Diseño

Principalmente se distinguen dos metodologías cuando se implementa un sistema empotrado, el co-diseño y el diseño basado en plataforma[2].

1. Co-Diseño: Esta basado en la cuidadosa determinación de la arquitectura del sistema, en la cual se hace el correspondiente particionamiento de las tareas realizadas en software y hardware. La principal fortaleza de la metodología de trabajo consiste en permitir el diseño concurrente, ya que una vez la arquitectura ha sido definida, los componentes software y hardware pueden ser diseñados, probados y optimizados separadamente; para finalmente ser integrados en un producto final, reduciendo así el tiempo de producción de un sistema completo.

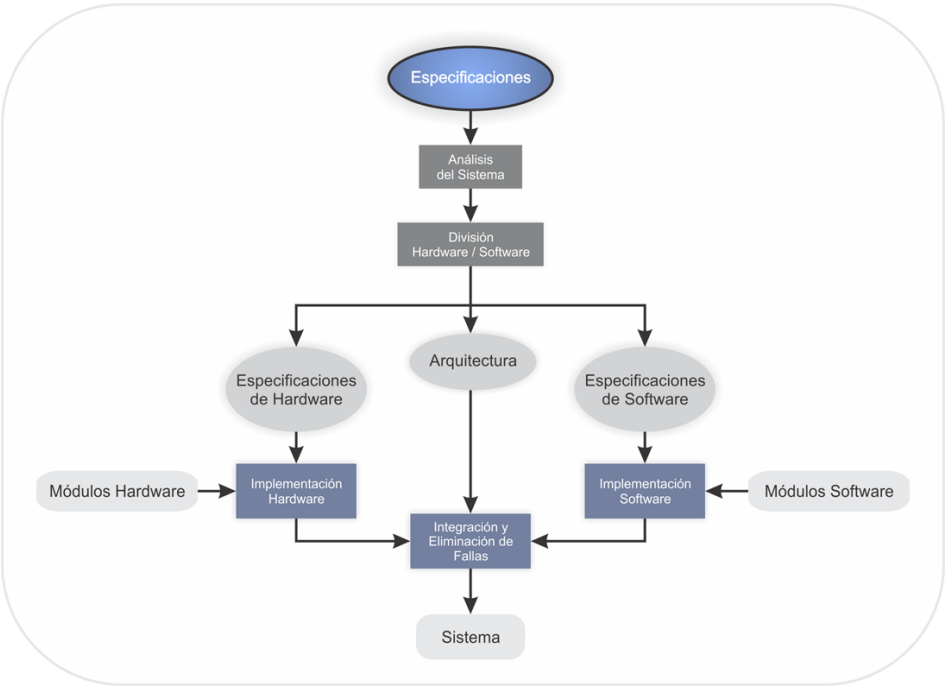

Figura 2.1: Flujo del Co-diseño

2. Basado en la Plataforma: Se enfoca en la selección de una plataforma, generalmente un SoC[2], que ejecutará la tarea que se necesite. Esta metodología esta compuesta por las siguientes fases:

   - Identificación y análisis de los requerimientos del sistema y modelos de software dentro de la plataforma harware.
   - Exploración de las diferentes alternativas de hardware dentro de la plataforma.

[2]Sistem on Chip

- Simulación de la arquitectura, para la evaluación y optimización.
- Desarrollo de diferentes capas de abstracción tanto en hardware como en software para la plataforma, tal como OS's[3], comunicación, librerías de aplicación, etc.

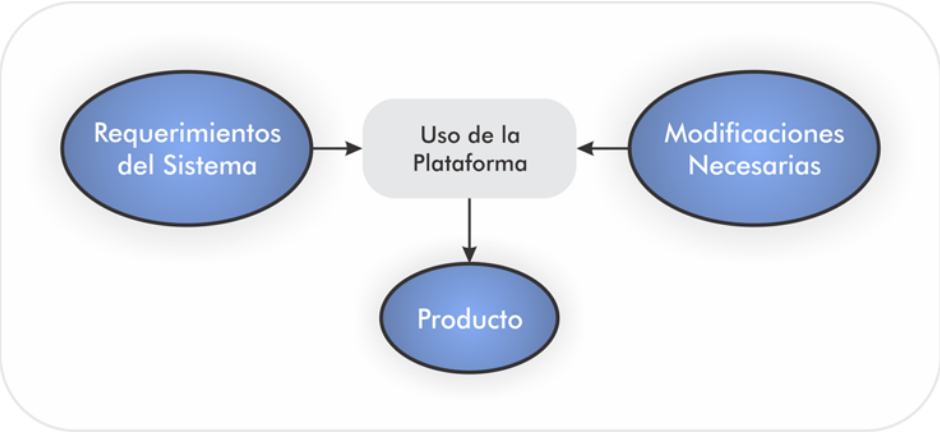

Figura 2.2: Flujo de Diseño Basado en Plataforma

En la gran mayoría de los diseños se necesita implementar estándares de alguna índole, adicionando otra variable a tener en cuenta en la aplicación de alguna de las metodologías nombradas anteriormente. La utilización de estándares provee beneficios, como la existencia de un SoC que implemente el estándar, conllevando perdida de control por parte del diseñador sobre la plataforma.
El diseñador debe sobrellevar los problemas de implementar el estándar ya que generalmente estos proveen unos requerimientos pero no especifican como deben ser implementados, aunque algunas veces se provee un diseño de referencia, que no siempre es útil. Siempre es recomendable seguir la documentación provista por los creadores del estándar así como experiencias de personas que han trabajado con el estándar.

2.3. Consideraciones Generales

El diseño de sistemas empotrados es una tarea que combina el desarrollo de hardware y software. La tarea inicial en el proceso corresponde a la selección de la metodología a utilizar y su correspondiente ejecución, para lo que se debe siempre poner en una balanza el tiempo de desarrollo, el costo del producto y la satisfacción de los requerimientos, dando a cada uno su peso correspondiente según las características de la aplicación en especifico.

[3]Sistemas operativos

Posteriormente se deben buscar experiencias de personas que hayan trabajado en productos similares, este tipo de información puede ser encontrada en las discusiones de grupos dedicados al desarrollo de sistemas para aplicaciones similares y algunas veces se pueden encontrar concejos útiles en la información provista por los fabricantes de microcontroladores.

Actualmente se debe tener en cuenta la gran dificultad existente para realizar un selección optima entre precio y desempeño, debido a que las hojas de datos generalmente no muestran los costos totales; por lo que en la actualidad, la meta de un diseñador de sistemas empotrados es *encontrar una solución aceptable en el tiempo especificado para finalizar el proyecto*[3], sin entrar en el error de intentar encontrar la mejor solución posible. Al igual que en otras industrias el objetivo es desarrollar un producto que trabaje apropiadamente y sea fácil de fabricar en el tiempo especificado.

En los sistemas empotrados, uno de los pilares fundamentales es la unidad de procesamiento[4] que permitirá la ejecución de las tareas en software, por lo que se debe tener cuidado al seleccionarlo. En la actualidad el precio de adquisición de las CPU's de 32 bits, ha centrado el desarrollo de sistemas empotrados en esta clase de dispositivos.

Algunos consejos útiles a la hora de seleccionar una unidad de procesamiento de 32 bits se presentan a continuación

- El dispositivo debe estar disponible para comprarse por unidad, vía online o por catalogo desde al menos uno de los mayores distribuidores siendo un anónimo para el distribuidor.
- Hojas de datos completas y disponibles sin acuerdos o comisión.
- Un sistema de desarrollo de bajo costo debe estar disponible.
- Debe existir contacto técnico directo con el fabricante desde la pagina del vendedor.
- El dispositivo debe haber sido despachado a OEMs[5] por un periodo mínimo de tres a seis meses.
- El Núcleo del microprocesador debe estar soportado por la cadena de herramientas GNU.
- Debe existir al menos un producto comercial distribuyéndose que use el dispositivo.

Al entrar mas a fondo en la selección de una CPU se debe evaluar si se utilizará un SoC comercial o si se realizará la construcción de la CPU mediante un PLD[6], lo cual será dado por la aplicación en especifico.

[4] CPU
[5] Original Equipment Manufacturers
[6] Programable logic Device

2.3.1. Selección de un SoC

A la hora de seleccionar un núcleo de procesamiento la única regla inviolable es la selección de un núcleo popular, para minimizar la carga de trabajo al generar el código especifico para el procesador. Hablando en forma general, en la actualidad existen seis núcleos de 32 bits ampliamente usados: Motorola 680xx, Intel X86, PowerPC, MIPS, SuperH y ARM.

Las arquitecturas basadas en la familia x86 tienen ventajas inmediatas:

- Es posible usar sistemas operativos compatibles de PC y herramientas de desarrollo libres.
- La instalación de sistemas operativos es simple.
- Existe facilidad para interactuar con periféricos.
- Existe el soporte para los drivers para gran cantidad de periféricos.
- Hay disponibilidad de gran cantidad de tarjetas madre con diferentes combinaciones de periféricos.

Conociendo las principales ventajas se deben conocer algunas desventajas:

- Núcleos basados en x86 son muy costosos.
- Consumo de potencia, calentamiento y dispositivos de gran tamaño.
- Es virtualmente imposible realizar una construcción manual del sistema.
- Los ICs[7] de periféricos para PC[8] tienen un periodo de producción muy corto, lo que posiblemente causará que el sistema sea obsoleto antes de tener ser usado por el usuario final.

El uso de sistemas basados en el núcleo x86 es recomendable si el objetivo del proyecto es tener un producto con una gran variedad de características en hardware sin gastar una gran cantidad de tiempo en la eliminación de fallos en el diseño de harware.

Pasando a los dispositivos RISC, tenemos plataformas como SuperH, la cual proporciona una gran familia de dispositivos útiles, MIPS parece ser ampliamente usado en ASICs[9] y ASSPs[10] de terceros. Los arquitectura PowerPC generalmente se encuentra en aplicaciones que demandan alto desempeño. Se ha encontrado que las plataformas de desarrollo para estos sistemas son costosas, no es fácil encontrar variedad en estas si no se demuestra un gran poder adquisitivo, pero son arquitecturas ampliamente soportadas en la actualidad.

[7] Circuito Integrado
[8] Computador Personal
[9] Aplication Specific Integrated Circuit
[10] Aplication Specific Standard Product

Como Recomendación de oro a la hora de utilizar un dispositivo RISC de 32 bits es trabajar con la arquitectura ARM, la cual presenta las siguientes ventajas, algunas son aplicables a las arquitecturas nombradas anteriormente:

- Es una arquitectura madura y muy conocida.
- Tiene muy buenas características de consumo de potencia en contra del desempeño.
- Variedad de características, seleccionables por el diseñador.
- Precios por unidad atractivos.
- ARM provee gran cantidad de diseños de referencia.
- Existe gran cantidad de software gratis de diferentes niveles de complejidad.

Se debe tener en cuenta que el núcleo ARM es para el mundo de los sistemas empotrados como el núcleo x86 es para el mundo de los computadores personales.

En general un IC que necesite circuitos analógicos complejos, diseño de PCB muy cuidadoso puede ser muy difícil de trabajar con una manufactura de bajo nivel. Al trabajar un chip de este tipo es recomendable invertir inicialmente en una tarjeta de desarrollo conocida antes de construir su propia PCB.

2.3.2. Selección de una FPGA

Mientras se esta evaluando diferentes chips para la aplicación, es posible verse tentado por SoC especializados ofrecidos por varias empresas manufactureras. Estos chips pueden tener interesantes periféricos específicos para varias aplicaciones.
Desafortunadamente, estos dispositivos son inalcanzables para los desarrolladores a pequeña escala, ya que para la adquisición se necesita realizar grandes pedidos o la firma de concesiones con los productores.
Algunas veces el camino a tomar es el trabajo con PLDs, que permiten mediante HDL[11] crear periféricos o procesadores en su totalidad[12]. Una recopilación de los SoftCores más populares en la actualidad con sus correspondientes características puede encontrarse en el Anexo A.

Existen PLDs de diferentes precios y capacidades, incluso algunos poseen procesadores ARM o PowerPC implementados en forma *Hard*[13]
El uso de la FPGA debe ser muy bien evaluado ya que puede aumentar bruscamente los costos o disminuirlos notablemente. Ya que el funcionamiento del circuito dentro de la FPGA depende principalmente del firmware que se escriba mediante HDL es posible llegar a concebir la posibilidad de construir un prototipo y realizar las pruebas

[11] Hardware Description Language
[12] SoftCores
[13] Implementados en silicio dentro del chip desde la fabrica

directamente sobre este, pero esto es solo recomendable si se ha trabajado anteriormente con estos dispositivos. Para personas iniciando en el mundo de los PLD lo mejor es adquirir una tarjeta de desarrollo.

2.4. Posibles vías a tomar en el ciclos de Desarrollo

- Si el código necesario puede ser desarrollado en una tarjeta de desarrollo fácilmente adquirible y económica , se debería usar como plataforma hardware para el proyecto.
- Si se esta construyendo un dispositivo de un solo uso, y se está completamente seguro de que no será necesario construir más de estas unidades y no se necesita construir alrededor algún dispositivo especifico, la vía a tomar es proponer una pieza de equipo de consumo con las características de hardware necesarias.
- Si se esta diseñando alrededor de un componente en especifico o combinación de componentes y la tarjeta de desarrollo es demasiado costosa o no es flexible para añadir los periféricos necesarios, la mejor opción es diseñar un circuito propio, creando unos cuantos prototipos en PCB[14], donde cada nuevo prototipo corregirá los errores encontrados en el anterior.

Si ninguno de de las opciones descritas anteriormente cumplen las expectativas, es recomendable que se desarrolle y se muestre el software en un *Embedded PC*[15] y luego reevaluar la utilización de alguna de las opciones anteriormente descritas.

2.5. Software Dentro de un Sistema Empotrado

En la actualidad el software que se encuentra dentro de un sistema empotrado ha aumentado su tamaño y complejidad debido al desarrollo de CPUs con más capacidades computacionales. Al incrementar el número de procesos ejecutados dentro de un dispositivo, es necesario tener una herramienta software encargada de administrar la ejecución de cada uno de los procesos, permitiendo al desarrollador concentrarse en generar el código de cada una de las tareas y no en la administración de estas.
Estas herramientas se conocen como SO[16] y fueron inventados para los PC[17] de propósito general, en los cuales se dispone de una gran cantidad de recursos y no se tienen restricciones de tiempo de ejecución de una tarea, situación que no ocurre en los sistemas empotrados, ya que allí se tienen recursos limitados, algunas veces precarios, y generalmente el proceso del cual esta encargado el sistema empotrado tiene estrictas restricciones en tiempos de ejecución, como lo es el control de frenado de un automóvil,

[14]Printed Circuit Board
[15]Computador personal empotrado
[16]Operating Systems
[17]Personal Computer

en el que si el sistema no realiza los ajustes pertinentes en el momento en que se detecta la perdida de estabilidad, la vida de una persona estaría corriendo peligro.
Para aplicaciones que requieren el cumplimiento a cabalidad de las restricciones de tiempo impuestas por el sistema y sea necesario un administrador de las tareas, se debió diseñar piezas especiales de software conocidas como RTOS[18] en las cuales se ofrece una variedad de APIs[19] que tienen la característica de poseer un tiempo de ejecución conocido e invariante ante la cantidad de tareas que se estén administrando.

Es importante resaltar que inicialmente los RTOS fueron diseñados para CPUs de alto desempeño pero en la actualidad se han ido optimizando para poder ser ejecutados por CPUs de bajo y alto desempeño sin presentar mayores diferencias en sus prestaciones [4], así como que la utilización de un RTOS no garantiza el cumplimiento de la tarea en un determinado tiempo, debido a que el tiempo de ejecución de las tareas administradas por el RTOS esta intrínsecamente ligado a la forma en que se describan por parte del programador, por loe que el programador debe conocer muy bien los límites y prestaciones del RTOS que utilice.

Debido a que el programador debe conocer la estructura de los RTOS, en la tabla 2.1 se muestran las principales diferencias con los SO de propósito general y en la tabla 2.2 se muestran las principales APIs. Como se muestra en el apéndiceB existen gran variedad de RTOS, pero cuando se construye un sistema empotrado con un RTOS, se debe seleccionar solamente uno. Existen diferentes criterios cuando se selecciona,

Característica	OS Genérico	RTOS
Scheduling	Tiempo compartido	Basado en prioridades
Tiempo en sincronización de tareas	Impredecible	Predecible
Comportamiento	No determinístico	Determinístico

Tabla 2.1: Principales diferencias entre RTOS y OS genéricos

principalmente se deberían tener en cuenta los siguientes[20]:

- Desempeño en tiempo real
- Soporte técnico
- Buenas herramientas software
- Compatibilidad software y hardware
- Costo total
- Documentación
- Capacidad de comunicación

[18]Real Time Operating System
[19]Application Programing Interfaces
[20]Tomado de: Influential factor in operating systems selection, embedded market survey, CMP, EE Times 2005, 2006 and 2007

API	Descripción
Manejo del sistema	Inicialización del OS, encendido y apagado del OS, bloqueo y desbloqueo de la CPU
Manejo de Interrupciones	Entrada y salida de funciones, comienzo y terminación de secciones criticas
Manejo de tareas	Creación, borrado, inicialización y terminación de tareas
Sincronización dependiente de tareas	Suspensión y reactivación de tareas
Comunicación y Sincronización	Semáforos, Colas, banderas de eventos, exclusiones mutuas, mensajes y almacenamiento de mensajes
Manejo de Memoria	Tamaño de memoria fijo o variable
Manejo de Tiempo	Obtención del tiempo del OS, timer del OS, etc.
Trace API	Rutina de hook dentro de algunas funciones del RTOS como es el scheduler

Tabla 2.2: Principales APIs provistas por un RTOS

- Reputación del distribuidor
- Derechos libres de utilización
- Tamaño de código, uso de memoria

Dentro del ítem, desempeño en tiempo real se pueden tener los siguientes criterios de selección[4]:

- Objetivos perseguidos por el RTOS
- Autor
- Esquema de Scheduling
- Soporte de lenguajes de programación
- Variedad de APIs
- Herramientas para depuración de errores

Como elemento de comparación es útil revisar los resultados de los Benchmarking realizados por empresas, especialistas en software y entusiastas, documentación técnica provista por los fabricantes de CPUs y estudios realizados por centros de investigación en el área de la computación.

2.6. Control Genérico de un Brazo Robótico de seis Grados de Libertad

La realización satisfactoria de una tarea por parte de un brazo robótico esta ligada al control cinemático y al control dinámico. El control cinemático esta comprendido por la generación de las trayectorias en base a alguna de las estrategias propuestas en la literatura, así como por el cálculo de los valores angulares de las articulaciones necesarios para describir la trayectoria, mediante la utilización de la cinemática directa

o inversa. Por otra parte el control dinámico es el encargado de que las articulaciones se encuentren posicionadas en los valores angulares provistos por el control cinemático, mediante la utilización de un controlador, por ejemplo un PID.

Cuando se desea llevar a la realidad cada uno de estos controles es importante tener en cuenta los cálculos que se deben realizar online y offline, ya que si se trabaja online debe existir un dispositivo que realice las operaciones necesarias en un tiempo determinado. Históricamente la mayor parte del control cinemático se ha realizado offline, ya que es el más exigente computacionalmente, debido a la cantidad de operaciones a realizar.
Dentro del control cinemático, el cálculo de los valores angulares de las articulaciones es particularmente atractivo al intentar llevarlo a una ejecución online, ya que al realizarse online brinda al robot la posibilidad de lograr una posición de forma autónoma, por lo que es conveniente conocer un poco más a cerca de los cálculos necesarios para obtener las variables angulares.

Los valores angulares de las articulaciones pueden ser calculados de dos formas. La primera, mediante la técnica conocida como cinemática directa, la cual consistente en la especificación detallada de los valores angulares de cada una de las articulaciones para lograr una posición, conllevando a la realización de varias pruebas, donde se hallarán analíticamente y experimentalmente la secuencia de valores a suministrar a los actuadores para causar el movimiento deseado.
La segunda, conocida como cinemática inversa, se basa en el cálculo de cada uno de los valores angulares necesarios para seguir una trayectoria y lograr una posición y orientación final, con solamente el suministro de algunos parámetros antes de empezar el movimiento.

Debido a la capacidad brindada por la cinemática inversa para implementar un sistema autónomo que calcule en tiempo real las variables necesarias para describir un movimiento, es importante conocer etapas, algoritmos y requerimientos operacionales que hacen posible la realización del movimiento. Conociendo estos parámetros es posible generar especificaciones para las plataformas hardware y software que estarán encargadas del procesamiento de los datos.
A continuación se muestran las etapas a llevar a cabo dentro del cálculo de las variables angulares por medio de la cinemática inversa:

- Establecimiento de la posición y orientación inicial de la pinza
- Establecimiento de la posición y orientación final de la pinza
- Establecimiento del tiempo de ejecución del movimiento
- Evaluación de realización del movimiento
- Cálculo de trayectorias a seguir

- Establecimiento de los tiempos requeridos en completar cada una de los puntos de paso de la trayectoria
- Cálculo de las variables angulares para lograr cada uno de los puntos de paso de la trayectoria .

Los valores angulares de las articulaciones pueden ser hallados por diversos métodos, tales como la transformación inversa, la matriz jacobiana inversa, el álgebra de tornillo, matrices duales, cuaterniones duales, interactivo y métodos geométricos, siendo este ultimo uno de los más populares, debido a su mayor carácter explicativo.

Al trabajar con alguno de los métodos geométricos es necesario realizar un desacople cinemático entre los tres grados de libertad superiores y los inferiores, debido a la complejidad matemática que implica considerar los seis grados de libertad acoplados cinemáticamente. Realizando la separación cinemática se tienen dos nuevas subetapas, la primera, que seria el cálculo de los ángulos de las tres articulaciones inferiores, con las cuales se logra la posición final, y como segundo se calcularía la solución para los tres grados de libertad superiores, logrando una orientación deseada[5], es importante resaltar que cada uno de los resultados de las subetapas es obtenido mediante la solución de ecuaciones trigonométricas y cuadráticas.

Si se utiliza otra técnica para resolver el problema cinemático inverso no es necesario desacoplar cinemáticamente las articulaciones, pero se debe tener en cuenta que los requerimientos computacionales se elevan, ya que por ejemplo al utilizar la matriz jacobiana inversa es necesario realizar la inversión de una matriz de 6x6 determinado número de veces, hasta llegar a una solución consistente, que brinde valores dentro de los rangos angulares de las articulaciones para lograr la posición y orientación deseada.

Capítulo 3

Desarrollo del Proyecto

Para el buen desarrollo del presente proyecto era necesario un amplio análisis de los requerimientos finales, así como del estado del arte de la tarjeta a implementar. Estos análisis llevan a un sin número de posibilidades o metodologías de desarrollo y evolución, donde las selecciones realizadas son la parte fundamental para encaminar el trabajo hacia un producto final. En el capítulo 2, se estableció que al diseñar prototipos estos pueden generarse mediante Co-Diseño o Basado en Plataforma. Igualmente se mencionó la importancia de basarse en los conjuntos de herramientas disponibles alrededor de las determinadas unidades de procesamiento, como lo son las tarjetas de desarrollo, las cadenas de herramientas y la información disponible sobre los dispositivos a usar.
Los conceptos traídos a colación anteriormente llevaron a usar principalmente la metodología de trabajo basada en una plataforma, tomando algunas ideas del co-diseño debido a que corresponde a la primera propuesta de la tarjeta de control para el robot MA2000, además que el equipo de desarrollo esta constituido por dos personas encargadas de la implementación, un asesor en el área de control y otro en el área de sistemas empotrados, así como que se debe cumplir con un plazo de cuatro meses para implementar el sistema y obtener los resultados de la primera propuesta de la tarjeta de control.

Debido a las razones anteriormente mencionadas, la metodología seleccionada constó de los siguientes pasos para el desarrollo del sistema:

1. Evaluación y análisis de requerimientos
2. Selección de CPU
3. Selección de Plataforma
4. Diseño del sistema basado en tarjeta de desarrollo
5. Pruebas

3.1. Evaluación Inicial y análisis de requerimientos

La principal motivación del presente proyecto es la puesta en funcionamiento del brazo robot MA2000, para que este pueda ser utilizado como herramienta académica, lo cual pone en evidencia la necesidad primordial de asegurar la reconfigurabilidad del sistema de control y de la plataforma de interacción con el brazo robot.
Igualmente se enfatiza el deseo de brindar una verdadera interacción entre los estudiantes y el sistema final, con lo cual se generaría la posibilidad de investigación en áreas afines a este, tales como control, automatización, sistemas empotrados, programación y robótica aplicada.
El sistema de control debe igualmente asegurar la mayor compatibilidad posible con los demás elementos del sistema, como lo son las tarjetas de potencia, la fuente de alimentación, los sensores y los actuadores del brazo robot.

El brazo robot MA2000 consta de seis grados de libertad otorgados por los movimientos de guiñada, alabeo y cabeceo del actuador, y las rotaciones en codo, hombro y cintura, como se puede observar en la Figura 3.1. La posición específica de cada uno de estos grados de libertad es sensada mediante un potenciómetro lineal, el cual varía un divisor de tensión. La proporción de tensión resultante debe ser digitalizada para ser interpretada. Esta digitalización se realiza mediante ADCs[1].

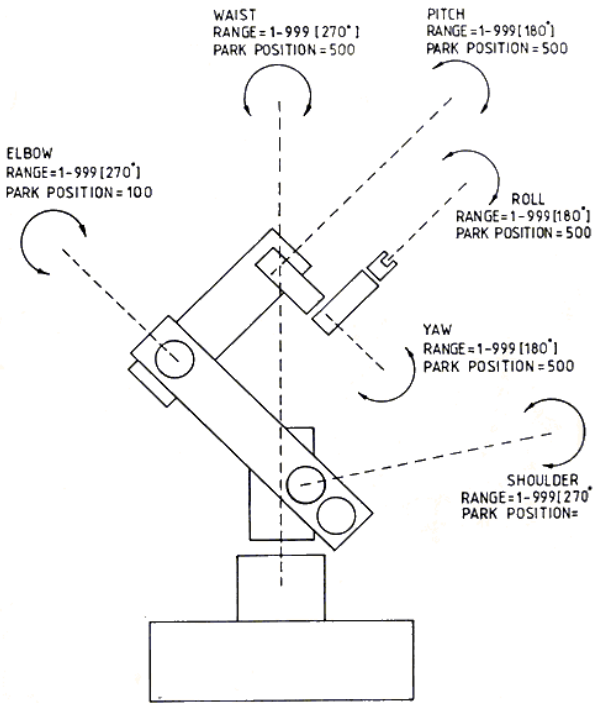

Figura 3.1: Grados de Libertad y Rangos de movimiento del MA2000 [1]

[1] Analog to Digital Converter

La tarjeta de control debe recibir comandos para la reubicación de cada uno de los actuadores en posiciones determinadas, con el ánimo de alcanzar una determinada ubicación espacial, lo cual es conocido como cinemática directa. Igualmente debe estar en capacidad de realizar los cálculos necesarios para determinar la trayectoria para llegar a una ubicación final, determinando las posiciones necesarias de los actuadores, conocido a su vez como cinemática inversa.

El origen de los comandos de cinemática directa o inversa puede ser local o remoto, lo cual implica en ambos casos el manejo de determinados protocolos de comunicación.

Para asegurar que los actuadores logren y conserven una posición deseada se requiere de un sistema de control. Este sistema de control puede ser clasificado como MIMO[2] si se consideran los efectos dinámicos entre actuadores, o como seis SISO[3] si se desacoplan estos efectos. Inicialmente se utilizará un esquema básico de control, con el uso de controladores PIDs, considerando los actuadores como seis sistemas SISO.

Finalmente todas las tareas de control, comunicación y cálculo de trayectorias deben realizarse cumpliendo con los parámetros de tiempo real impuesto por la respuesta del brazo MA2000. Este requisito se puede lograr al hacer uso de RTOS que faciliten el control de dichas tareas y administren los recursos que estas demandan.

3.2. Elección de CPU para desarrollo de la tarjeta

Con base en el sistema final deseado, descrito a grandes rasgos anteriormente, se presentan las diferentes CPUs con que se podría implementar:

- Programmable Logic Controller (PLC).
- Microcontrolador.
- Microprocesador.
- Computador Empotrado.
- Programmable Gate Array (FPGA) con SoftCore.

En la tabla 3.1 se presenta una comparación respecto al cumplimiento de los principales requerimientos por parte de las CPUs, para la implementación de la tarjeta de control. Priorizando la reconfigurabilidad del sistema y la posibilidad de interacción en futuros trabajos de investigación es evidente la ventaja que presenta trabajar con una FPGA con SoftCore[referenciar artículo HardCore Vs Softcore], al momento de la generación de prototipos. Sin embargo se debe tener en cuenta que al trabajar con FPGAs se deben tener elementos externos que realicen tareas que no pueden realizar las FPGAs, como por ejemplo la captura de señales analógicas de alta frecuencia.

Debido a que se eligió trabajar con una FPGA, se debía elegir un SoftCore para ser sintetizado dentro de la FPGA, para lo cual se realizó una búsqueda de los SoftCores

[2]Multiple Input, Multiple Output
[3]Single Input, Single Output

Unidad Central de Procesamiento	Reconfigurabilidad		Interactividad	Capacidad Computo	Soporte			
	Software	Hardware			ADC	PID	RTOS	Protocolos Comunicación
PLC	Media	Muy Baja	Baja	Alta	Si	Si	Bajo	Medio
Microcontrolador	Alta	Baja	Alta	Alta	Si	Si*	Alto	Medio
Microprocesador	Alta	Baja	Media	Alta	Si	Si*	Alto	Medio
Computador Empotrado	Muy Alta	Muy Baja	Media	Muy Alta	No	Si*	Alto	Alto
FPGA con SoftCore	Alta	Muy Alta	Alta	Media*	Si**	Si**	Medio	Alto**

*Parámetro dependiente de la reconfigurabilidad Software.
**Parámetro dependiente de la reconfigurabilidad Hardware.

Tabla 3.1: Comparación Unidades Centrales de Procesamiento

más populares en el mercado, dando como resultado la información mostrada en la sección A.

Luego de conocer los SoftCores más populares en el mercado, se procedió a ver cual de estos era adecuado. Teniendo en cuenta que los criterios mostrados en la sección 2.3.1, se llegó a la conclusión de que la CPU debía cumplir los siguientes requerimientos:

- Trabajar con registros de 32 bits
- Manejar FPU
- Ejecutar una instrucción por ciclo
- Presentar configurabilidad dentro de su estructura
- Tener Cadena de Herramientas para Compilación de software soportada por GNU
- Tener Documentación clara y concisa
- Existencia de diseños previos basados en la CPU
- Software que automatice el proceso de Adecuación de la Plataforma
- Facilidad de Adquisición.

Con los requerimientos antes mencionados se debe elegir entre los SoftCores LEON3, LEON2, OpenRisc 1200 y MicroBlaze. El SoftCore Nios II ha sido retirado de la comparación ya que la utilización de este SoftCore, implicaría el tramite y adquisición de Licencias para el uso de este SoftCore y para las herramientas necesarias a diferencia de MicroBlaze. La E3T posee las licencias que permiten la utilización del SoftCore y la totalidad de las herramientas software provistas por Xilinx.

Dentro de las posibles opciones se optó por trabajar con MicroBlaze ya que se le dio mas peso a la experiencia en la utilización del SoftCore, existencia de diseños previos realizados por integrantes de la comunidad E3T y ventajas ofrecidas por la cadena de herramientas del dispositivo.

3.3. Selección de Plataforma

Luego de haber seleccionado una CPU adecuada, se realizó una evaluación de las tarjetas de desarrollo accesibles por la E3T, mostradas en la tabla 3.2, que brindarán los elementos necesarios para trabajar con FPGA y SoftCore. Debido a la amplia experiencia que se posee dentro del campus universitario en el trabajo con dispositivos de la empresa Xilinx y las pocas diferencias entre las FPGAs provistas por los diferentes fabricantes desde el punto de vista del programador, se optó por trabajar con dispositivos de la familia Xilinx.
Tomando como base de comparación las tarjetas de desarrollo para FPGAs y SoftCore ofrecidas por la empresa Xilinx[4], se llegó a la conclusión de que el material de trabajo disponible por la E3T provee una amplia gama de posibilidades de selección de plataformas que se encuentran vigentes en el mercado actual, llevando a la utilización de una de estas tarjetas de desarrollo como plataforma del proyecto.

Plataforma	Cantidad
Spartan 2	5
Spartan 3E	10
Spartan 3A DSP	4
Spartan 3AN	10
Virtex 5	2

Tabla 3.2: Tarjetas de Desarrollo accesibles por la E3T

Teniendo en cuenta los requerimientos del sistema y la posibilidad de generar un producto terminado, se puede hacer un primer acercamiento a la tarjeta de control final, mediante la utilización de la FPGA xcs700anfgg484, perteneciente a la familia Spartan 3AN, que es una FPGA de gama media dentro de las provistas por la empresa Xilinx. Esta FPGA posee la capacidad de albergar un sistema de control empotrado, mediante un circuito combinacional o mediante un SoftCore. Además puede almacenar archivos de configuración en memoria no volátil y puede ser producida en masa a costos razonables, ya que el precio por unidad esta en 37.14 USD[5]

La tarjeta de desarrollo *StarterKit 3AN*,que se puede observar en la figura 3.2, contempla un módulo para conectividad Ethernet lo cual aumenta las posibilidades para la obtención de comandos de manera remota y local. Igualmente dispone de múltiples periféricos que pueden ser utilizados en etapas posteriores a este proyecto para mejorar la interactividad y conectividad, tales como puertos Seriales RS232, puertos VGA, puerto PS2, memoria DDR2, etc.

[4]http://www.xilinx.com/products/boards_kits embedded.htm
[5]valor consultado en DigiKey Corp. el 23 de enero de 2011

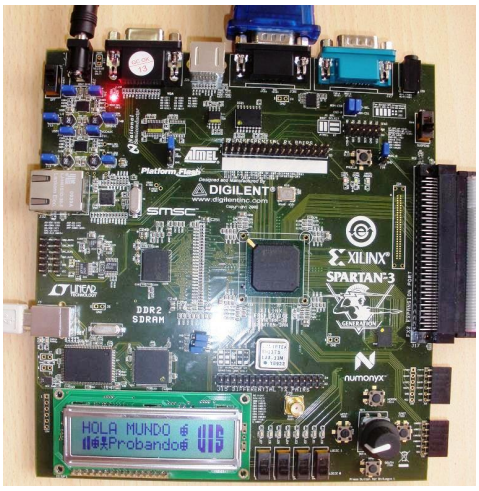

Figura 3.2: Spartan 3AN Starter Kit

3.4. Módulo PID

El controlador PID[6] es uno de los controladores con realimentación más comunes utilizados con sistemas dinámicos, gracias a su gran versatilidad y la facilidad de su diseño, asegurando un control sólido. El diversificado uso de este tipo de controlador es evidente pues su implementación se ha dado desde sistemas analógicos, como los neumáticos y mecánicos, hasta sistemas digitales, como en microprocesadores y de manera cada vez más importante en FPGA.

Al disponer de una plataforma de desarrollo basada en una FPGA no solo se obtiene la posibilidad de implementar estrategias de control en Software utilizando un SoftCore, de igual forma que se haría en un microprocesador, microcontrolador y demás, sino que estas estrategias pueden constituirse con hardware dedicado construido con los recursos propios de la FPGA.

Con en ánimo de proporcionar una mayor versatilidad a la plataforma de control desarrollada en el presente proyecto, se decidió dedicar hardware de la FPGA para la implementación de un controlar PID para cada uno de los seis grados de libertad de los que goza el brazo manipulador objetivo de esta plataforma.

El algortimo de un controlador PID analógico básico, esta dado por la Equación3.1, donde $u(t)$ es la señal de control, $e(t)$ es la señal de error de comparar una consigna $u_c(t)$ y la señal realimentada $y(t)$ y K, T_i y T_d corresponden a constantes de control.

$$u(t) = K * \left(e(t) + \frac{1}{T_i} * \int_0^t e(\tau)\,\mathrm{d}\tau + T_d * \frac{\mathrm{d}(e(t))}{\mathrm{d}t} \right) \tag{3.1}$$

Sin embargo esta primera aproximación presenta algunos inconvenientes tales como la amplificación de ruido por parte del término derivativo puro y la posibilidad de saturación del actuador ante la señal de control puede causar un desborde del integrador

[6]Proporcional-Integral-Derivativo

llevando a obtener una respuesta transitoria lenta, entre otros. Otro aspecto a tener en cuenta, es que la implementación digital de un controlador PID implica un muestreo periódico, y por tanto el algoritmo debe analizarse de manera discretizada. Otro efecto de un controlador digital reside en la perdida de precisión implicita al adquirir las señales de realimentación mediante un circuito ADC y a la representación en matemática binaria de las variables de control. Esta perdida de presición afecta principalmente la parte proporcional del controlador PID y por tanto es una buena práctica permitir que solo una parte de la consigna actue en esta.

El algoritmo discreto a implementar mediante hadware en la FPGA se presenta en la Ecuación 3.2. En esta ecuación T es el período de muestreo, N es una constante de control para el término derivativo, b es la constante de fraccion para la consigna, y k denota el k-ésimo instante de muestreo.

$$\begin{aligned} u(kt) &= P(kT) + I(kT) + D(kT) \\ P(kT) &= K\left(b * u_c(kT) - y(kT)\right) \\ I(kT) &= I((k-1)T) + \frac{KT}{T_i}\left(u_c((k-1)T) - y((k-1)T)\right) \\ D(kT) &= \frac{T_d}{T_d + NT} * D((k-1)T) - \frac{KT_dN}{T_d + NT}\left(y(kT) - y((k-1)T)\right) \end{aligned} \tag{3.2}$$

Implementar directamente la Ecuación 3.2 requiere 5 multiplicadores, 7 sumadores y 4 registros de retraso [6]. Ya que este mismo esquema se usaría para controlar los seis grados de libertad del manipulador y la FPGA tiene recursos limitados, no es óptimo. Para solventar este problema se hace uso de la interpretación de la multiplicación como una Aritmética Distribuida (DA, por sus siglas en ingles), donde se aprovecha la representación binaria de los números a multiplicar en un esquema basado en LUT (Look-Up Table), componentes básicos de una FPGA. Explicaciones detalladas del principio matemático de la DA puede ser consultado en [6, 7, 8].

Aplicando la teoría de DA siendo $u_c(kT)$, $u_c((k-1)T)$, $y(kT)$, $y((k-1)T)$ representados por N bits en las ecuaciones 3.2, obtenemos

$$\begin{aligned} P(kT) &= \sum_{j=0}^{N-1} K\left(b * u_c(kT)[j] - y(kT)[j]\right) * 2^j \\ I(kT) &= \sum_{j=0}^{N-1} \left(I((k-1)T)[j] + \frac{KT}{T_i}\left(u_c((k-1)T)[j] - y((k-1)T)[j]\right)\right) * 2^j \\ D(kT) &= \sum_{j=0}^{N-1} \frac{T_d}{T_d + NT}\left(D((k-1)T)[j] - KN\left(y(kT)[j] - y((k-1)T)[j]\right)\right) * 2^j \end{aligned} \tag{3.3}$$

De las ecuaciones 3.3 se obtienen las LUT que se presentan en las tablas 3.3, 3.4 y 3.5, las cuales serán implementadas mediante hardware en la FPGA.

$u_c(kT)[j]$	$y(kT)[j]$	LUT_P
0	0	0
0	1	$-K$
1	0	Kb
1	1	$Kb-K$

Tabla 3.3: LUT para Proporcional

$I((k-1)T)[j]$	$u_c((k-1)T)[j]$	$y((k-1)T)[j]$	LUT_I
0	0	0	0
0	0	1	$-(KT/T_i)$
0	1	0	KT/T_i
0	1	1	0
1	0	0	1
1	0	1	$1-(KT/T_i)$
1	1	0	$1+(KT/T_i)$
1	1	1	1

Tabla 3.4: LUT para Integral

$D((k-1)T)[j]$	$y(kT)[j]$	$y((k-1)T)[j]$	LUT_D
0	0	0	0
0	0	1	$KT_dN/(T_d+NT)$
0	1	0	$-KT_dN/(T_d+NT)$
0	1	1	0
1	0	0	$T_d/(T_d+NT)$
1	0	1	$(1+KN)T_d/(T_d+NT)$
1	1	0	$(1-KN)T_d/(T_d+NT)$
1	1	1	$T_d/(T_d+NT)$

Tabla 3.5: LUT para Diferencial

Como se muestra en las Ecuaciones 3.3, para el calculo de $P(kT)$, $I(kT)$ y $D(kT)$ se toma el j-ésimo bit de $u_c(kT)$, $u_c((k-1)T)$, $y(kT)$, $y((k-1)T)$, $I((k-1)T)$ y/o $D((k-1)T)$ según corresponda, como entradas de una determinada LUT, y su correspondiente salida pasa a ser multiplicada por 2^j, que en números binarios corresponde a un desplazamiento por j bits hacia la izquierda, y finalmente este valor se acumula. La operación se realiza para el intervalo de $j \in [0, N-1]$ y el valor acumulado corresponde a $P(kT)$, $I(kT)$ ó $D(kT)$ segun sea el caso.

Algunas de las consideraciones generales para la implementación de la lógica interna del controlador PID con esquema DA se presentan a continuación:

- Las señales de entrada a las LUT pueden ser con representación sin signo o con representación en complemento a dos.
- Debido a las operaciones implicitas en las constantes de salida de las LUT estas se manejaran en complemento a dos.
- Las longitudes en bits de las señales de entrada a las LUT pueden ser distintas, pero j tomará el rango de la señal de mayor longitud. Si por ejemplo en la LUT derivativa $y(kT)$ y $y((k-1)T)$ son de M bits pero $D((k-1)t)$ es de N bits, donde $M < N$, entonces $j \in [0, N-1]$.
- Al trabajar señales de entrada a las LUT con diferentes longitudes de representacion en bits, se debe contemplar la necesidad de extender el bit de signo. Siguiendo el ejemplo anterior, si $y(kT)$ es siempre positiva, $y(kT)[j] = 0$ y $y((k-1)T)[j] = 0$ al evaluar en $j \in [M, N-1]$, pero si $y(kT)$ es representada en complemento a dos entonces $y(kT)[j] = MSB$ y $y((k-1)T)[j] = MSB$ para el mismo intervalo de j, donde el MSB (Most Significant Bit) es el bit de signo del complemento a dos. Figura 3.3

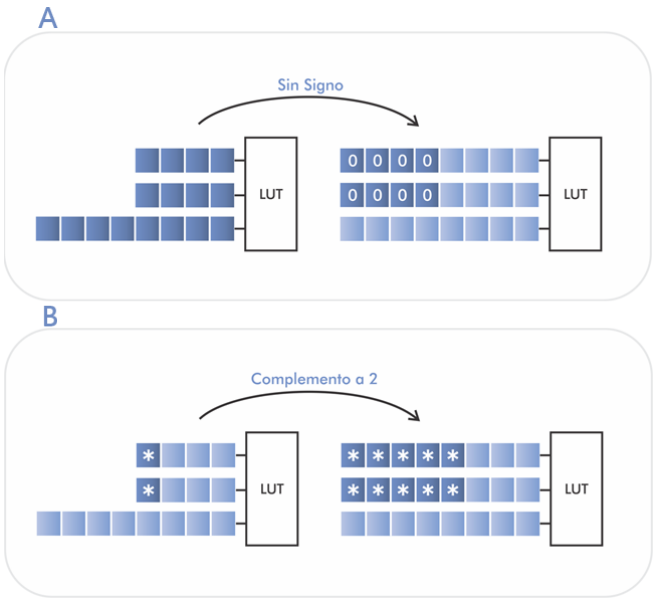

Figura 3.3: Extensión del Bit de Signo

- Para representar valores decimales sin utilizar representación en punto flotante, se suelen asignar una determinada cantidad de bits para las potencias negativas de dos, lo cual es conocido como representación en punto fijo.

- Adicionalmente a la determinación de la precisión decimal a manejar en punto fijo, se debe tambien analizar el hecho de que al realizar la multiplicación y conservar todos los bits del resultado, la posición de la coma decimal del resultado se verá alterada, siendo ésta la suma de las posiciones decimales de los multiplicandos. Por lo anterior, si el resultado del acumulador tendrá la misma presición decimal de sus antecesores, se deberán eliminar los X bits menos significativos que excedan la presición decimal deseada. Figura 3.4

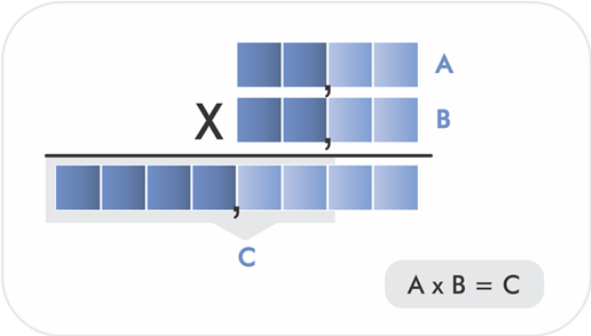

Figura 3.4: Corrimiento de la posición decimal. A, B y C tienen igual presición decimal.

- Si las señales de entrada a las LUT representan valores decimales mediante punto fijo, estas deberan ser alineadas por la posición de la coma decimal, agregando bits cero por derecha de ser necesario.

- Generalmente al aplicar DA todos los parámetros de control son conocidos y fijos, por lo que las valores de salida de las LUT pueden ser precalculados y sintetizados en hardware. Sin embargo al preveer la necesidad de flexibilidad en los parámetros de control, se recargará la tarea del calculo y asignación inicial de valores de salida de las LUT a la Unidad Centralizada de Procesamiento, en este caso un SoftCore, mediante registros.

El esquema general del manejo de datos (Datapath) de la implementación del controlador PID mediante Aritmética Distribuida se presenta en la Figura 3.5

La lógica de la Figura 3.5 , donde no se muestran las conexiones de reloj y reinicio por simplicidad, es controlada con una pequeña maquina de estados, cuyo diagrama se muestra en la Figura 3.6

3.5. Módulo PWM

Este módulo permite en un sistema de control, la transformación de la información contenida en los datos de n bits provistos por el controlador a una señal de 1 bit, que posee las características de una señal analógica y que será fácilmente aplicable a los actuadores del sistema a controlar.Una de las principales ventajas que ofrece la

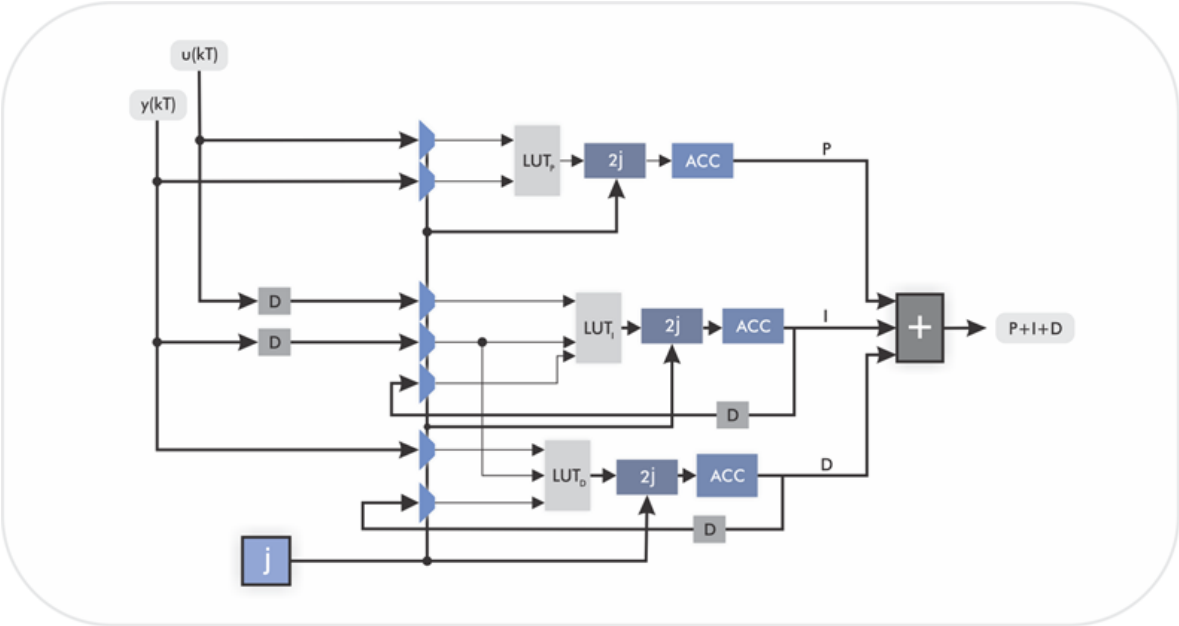

Figura 3.5: Datapath de Controlador PID con DA

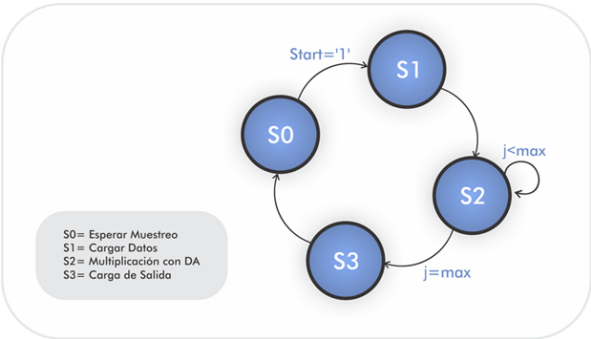

Figura 3.6: Diagrama de Estados para el control del PID con DA

utilización de señales PWM en los sistemas de control es que pueden ser generadas por circuitos digitales, quitando la necesidad de utilizar un DAC[7], reduciendo la complejidad y los costos del sistema.

Debido a que el sistema manejará 6 motores de DC simultáneamente, es necesario tener 6 dispositivos idénticos corriendo simultaneamente dentro del sistema de control. Para lograr este fin se plantea la generación de un módulo PWM descrito en VHDL, con lo que se obtiene control total sobre cada uno de las características de un módulo PWM, así como un alto nivel de repetibilidad que brindan los módulos VHDL.

El circuito digital de un módulo PWM genérico se puede presentar como se muestra en la figura 3.7, este circuito digital es de fácil implementación pero debido a que

[7]Digital to Analog Converter

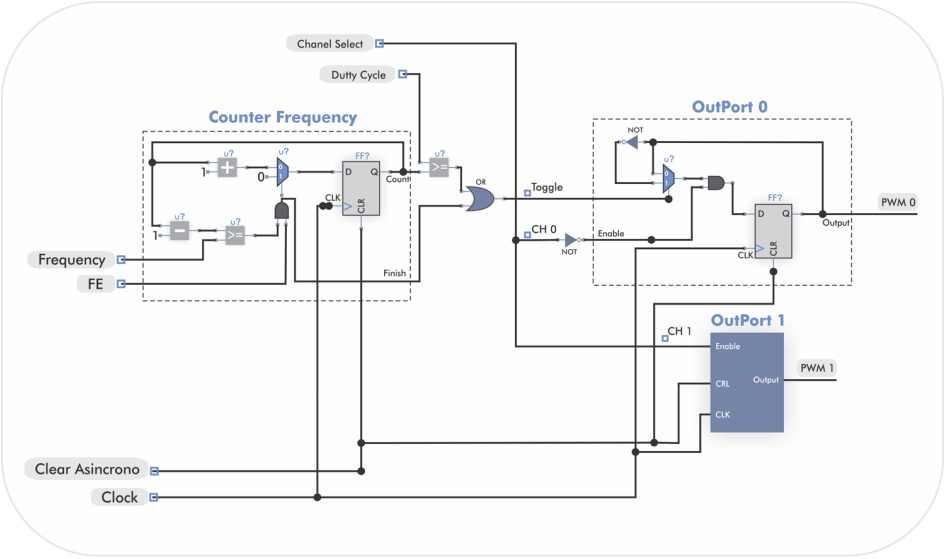

Figura 3.7: Esquema PWM Básico

su estructura es genérica, implica una adecuación en software para cada una de las aplicaciones.

Cuando se utiliza un módulo PWM para controlar un puente H que maneja motores de DC, es de vital importancia proveer un tiempo muerto al cambiar el sentido de giro del motor, ya que el tiempo de extinción de la corriente de las ramas no es cero. Normalmente esta tarea se realiza en software, porque se trabaja con módulos PWM genéricos, pero debido a que en esta aplicación se implementará el módulo, se decide descargar al procesador de esta tarea e incluirla en hardware, disminuyendo un tiempo muerto en el algoritmo de control ejecutado por el procesador. Para proveer esta característica en hardware, se proponen el siguiente esquema mostrado en la figura 3.8.

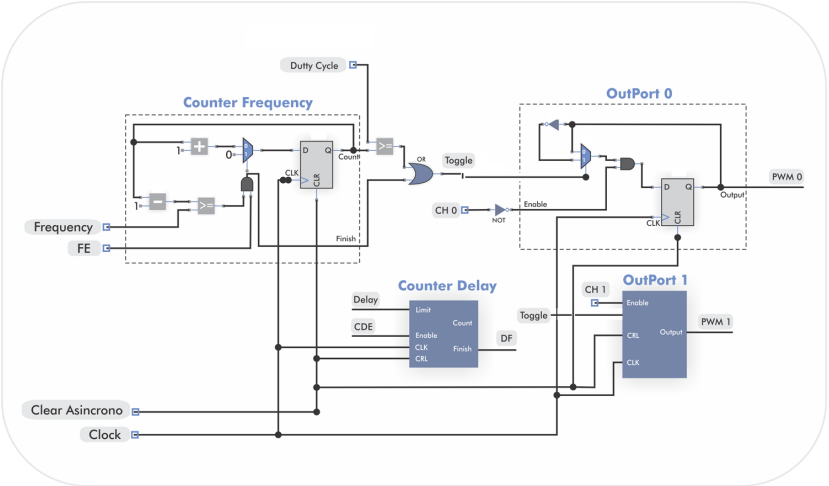

Figura 3.8: Datapath PWM

En el diseño presentado anteriormente se utiliza una FSM[8] que administra el flujo de datos entre la unidad PWM y cada uno de los puertos, presentando la posibilidad de lograr un retardo adicional a la hora de cambiar de canal. El diagrama de estados de la FSM implementada se muestra a continuación:

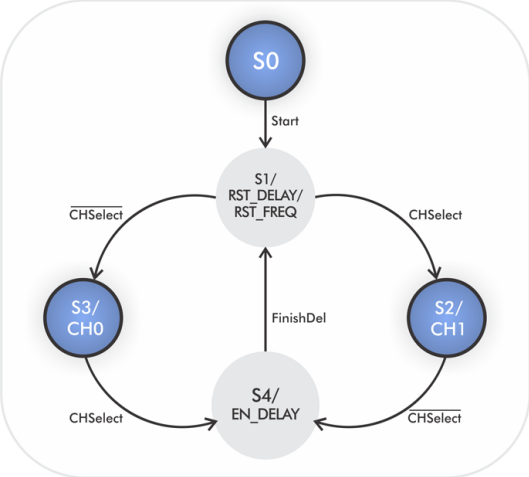

Figura 3.9: FSM para Control de PWM

3.6. Tarjeta de Interfaz con Puma MA2000

El objetivo de esta tarjeta era brindar un medio de comunicación entre los potenciómetros que proveen las posiciones angulares de cada una de las articulaciones y el interior de la FPGA. Debido a que se deseaba conservar la mayor parte del robot se debió realizar una recopilación de información sobre la forma en que estaba interconectado el robot PUMA MA2000 de la empresa TecQuipment.

Iniacialmente se optó por contactar a la empresa TecQuipment para solicitar soporte sobre el robot, pero la respuesta fue que el robot se encontraba descontinuado, pero que podían suministrar los planos de los circuitos eléctricos del robot,que se pueden encontrar dentro de la carpeta *Tequiment_info*, adjunta con el documento, lo cual aceleró el proceso de identificación de las conexiones.

Posteriormente a la identificación del plano se llegó a que se debían respetar las siguientes conexiones:

- Conector de 26 pines: Encargado de contener las 16 señales de PWM que irán a la tarjeta de drivers y 10 conexiones de propósito general, que en el diseño antiguo eran usadas para canalizar señales que controlaban relés usados para operaciones de emergencia.

[8]Finite State Machine

- Conector de 16 pines: provee vía a las señales provenientes de los potenciómetros hacia la FPGA, brinda conexiones a estas señales para realizar mediciones y provee alimentación para los potenciómetros.

Adicionalmente se identificó que era necesario proveer un ADC que se encargara de entregar las señales de los 6 potenciometros en un digital a la FPGA, además de que se debía proveer alimentación a los 6 potenciómetros desde una fuente externa.

Para proveer la alimentación a los potenciometros se optó por conservar la convención de alimentación provista por la empresa TecQuipment, que correspondía a 10 V para alimentar 6 potenciómetros de 2 $k\Omega$, con lo que se puede tener un presupuesto de las variables eléctricas nominales esperadas, mostrado en la tabla 3.6. Como los potenciómetros deben ser alimentados por una fuente externa se plantea utilizar la fuente de alimentación provista por el convertidor AC/DC provisto por la tarjeta *Spartan 3AN Starter Kit*, junto con un doblador de tensión. La decisión se sustenta en base a que utilizar estos dos elementos, se reduce el número de elementos a utilizar, área utilizada y disminución de costos.

Para la implementación de esta etapa se tubo en cuenta que el convertidor AC/DC era capaz de suministrar 3.2 A, 5 V y la tarjeta *Spartan 3AN Starter Kit* tiene un consumo máximo de 2 A a 5V lo que garantiza que la fuente es capaz de satisfacer los requerimientos mostrados en la atabla 3.6 y los de la tarjeta *Spartan 3AN Starter Kit*. Al utilizar el convertidor AC/DC provisto en la tarjeta de desarrollo se posee el problema de que se tiene una tensión de salida de 10 V y solo se poseen 5V, para lo que se utilizó un regulador de tensión doblador e inversor de tensión referencia LT1026, el cual cumple con los requerimientos provistos en la tabla 3.6.

Parámetro	Valor
Tensión	10 V
Corriente por potenciómetro	5 mA
Corriente Total	30 mA
Potencia por potenciómetro	50 mW
Potencia Total	300 mW

Tabla 3.6: Demanda de Potencia Potenciometros

Como ultimo se revisó los requerimientos del ADC, pero debido a que es parte fundamental en el proceso de control, se hablará detalladamente de él en una sección posterior.

3.7. Adquisición de Realimentación mediante ADC

Con el fin de controlar la posición de cada uno de los grados de libertad del brazo robot, es necesario sensar las señales provenientes de los potenciometros acoplados a

cada uno de estos. Un circuito conversor Analógico a Digital, o ADC, es entonces la solución clara para realizar la interfaz entre estas señales y la circuitería lógica de la FPGA.

Dependiendo del esquema de control a implementar, y de la necesidad de presición se deben establecer algunos de los parámetros de selección básicos de un ADC:

- Número de Bits para la representación de la señal analógica.
- Rango de señal de entrada adecuado a la excursión total de la señal analógica.
- Taza de muestreo.
- Arquitectura interna.
- Muestreo periódico o bajo comando.
- Número de canales de entrada de señal analógica.
- Muestreo Simultaneo o Multiplexado.
- Protocolo de Transmisión de Datos.
- Manejo de Reloj Externo o Interno.
- Requerimientos de referencia de tensión.

Dadas las características de la aplicación final del presente proyecto se tomó en cuenta la selección de un ADC con 12 bits de resolución, muestreo comandado, muestreo simultaneo, seis o más canales analógicos de entrada y un muestreo de mínimo 10KSPS[9], como parámetros principales. La arquitectura interna para los anteriores requerimientos, dada la velocidad requerida y la resolución, que más se maneja comercialmente es la SAR[10]. La simultaneidad en el muestreo de los diferentes canales se consideró un factor importante, pues de esta manera se cumple con la teoría básica del control que contempla muestreos en igual instante de tiempo para todas las señales de entrada, en orden de asegurar estabilidad y predictibilidad.

Es de tener en cuenta que la tarjeta de desarrollo para la Spartan-3AN, en que esta basada la plataforma del presente proyecto, posee un ADC de dos canales de entrada multiplexados en tiempo, de 14 bits de resolución. Este ADC podría ser utilizado pero debería añadirse circuitería para multiplexar las seis señales de entrada a las dos disponibles en este; además de que esto incurriria en una menor velocidad de muestreo y se perdería la nocion de la teoría de control al desear controlar con tiempos de muestreo menores.

En la Tabla **??** se muestran algunos de los ADCs revisados y que cumplen con las especificaciones planteadas para su seleccion final.

El ADC escogido es el ADS8558[11] fabricado por Texas Instruments, el cual además de cumplir a cabalidad los parámetros deseados, presenta funcionalidades extras como:

[9]Kilo-Samples Per Second, es decir, Mil Muestras por Segundo

[10]Succesive Aproximation Register, es decir, Registro de Aproximación Sucesiva

[11]Información del fabricante y Datasheet disponible en http://focus.ti.com/docs/prod/folders/print/ads8558.html

Tabla 3.7: LUT para Diferencial

No	FABRICANTE	REFERENCIA	VELOCIDAD	RESOLUCION	ARQUITECTURA	CANALES	COMUNICACIÓN		SIMULTANEO
1	Texas Instruments	ADS8558	730kSPS	12 bits	SAR	6	Parallel	SPI	Si
2	Texas Instruments	ADS7828	50 kSPS	12 bits	SAR	8	I2C	—	No
3	Texas Instruments	ADS1178	52 kSPS	16 bits	Delta-Sigma	8	SPI	FS	Si
4	National Semiconductor	ADC128S022	200 kSPS	12 bits	SAR	8	SPI	QSPI	No
5	National Semiconductor	ADC12048	216 kHz	12 bits	SAR	8	Parallel	—	No
6	National Semiconductor	LM12H458	140k	13 bits	SAR	8	Parallel	—	No
7	Analog Devices	AD7861	28,6 kSPS	11 bits	SAR	7	Parallel	—	No
8	Analog Devices	AD7658	250 kSPS	12 bits	SAR	6	Parallel	SPI	Si
9	Analog Devices	AD7606-6	200 KSPS	16 bits	SAR	6	Parallel	SPI	Si
10	Analog Devices	AD7859	200 kSPS	12 bits	SAR	8	Parallel	Byte	No
11	Linear Technologies	LTC1853	400 ksps	12 bits	SAR	8	Parallel	—	No
12	Linear Technologies	LTC1851	1.25Msps	12 bits	SAR	8	Parallel	—	No

Configurabilidad por Hardware o Software, Transmisión de Datos por Palabras o por Bytes, posibilidad de seleccion entre un reloj de conversión interno o uno externo, posibilidad de seleccion entre referencia de tensión interna o externa, configurabilidad del rango admisible de entrada de tensión analógica y transmisión de datos mediante protocolo Paralelo o Serial. Este ADC es además compatible con el ADS8557 y ADS8556, los cuales manejan 14 y 16 bits de resolución respectivamente, lo cual permitiría la migración a una mayor precisión sin mayor modificación de Hardware o Software.

Para el presente proyecto el ADS8558 se manejará con transmisión de datos en formato palabra de manera paralela, aprovechando nuevamente las capacidades lógicas de la FPGA, y con configuración por Hardware.

Utilizando Xilinx ISE Design Suite se implementó la comunicación paralela con el ADS8558. El protocolo de adquisición de datos mediante el ADC se puede describir mediante los siguientes pasos:

1. Se espera el comando por parte del usuario (en este caso el SoftCore) para iniciar.
2. Se ponen en alto los pines $CONVST_A$, $CONVST_B$ y $CONVST_C$ para que se inicie la conversión interna de los seis canales del ADC.
3. El módulo espera a que se ponga en bajo el pin $BUSY$ del ADC, el cual indica que la conversión se encuantra en proceso.
4. Se ponen en bajo las señal de seleccion del chip, ó $\bar{CS}$, y de inicio de conversión.
5. Se espera un poco más al tiempo mínimo definido y se pone en bajo el pin de lectura, o $\bar{RD}$.
6. Se lee de manera paralela los pines desde DB0 hasta DB15 y se guardan en un registro.
7. Se pone en alto la señal RD y se repiten los pasos 4, 5 y 6, las seis veces necesarias para obtener la lectura de cada canal.

8. Finalmente se retornan CS a Alto.

Cada uno de los pasos anteriores tiene unas restricciones de tiempo especificadas en la hoja de datos del ADS8558. Un esquema general del protocolo paralelo a manejar se presenta en la Figura 3.10

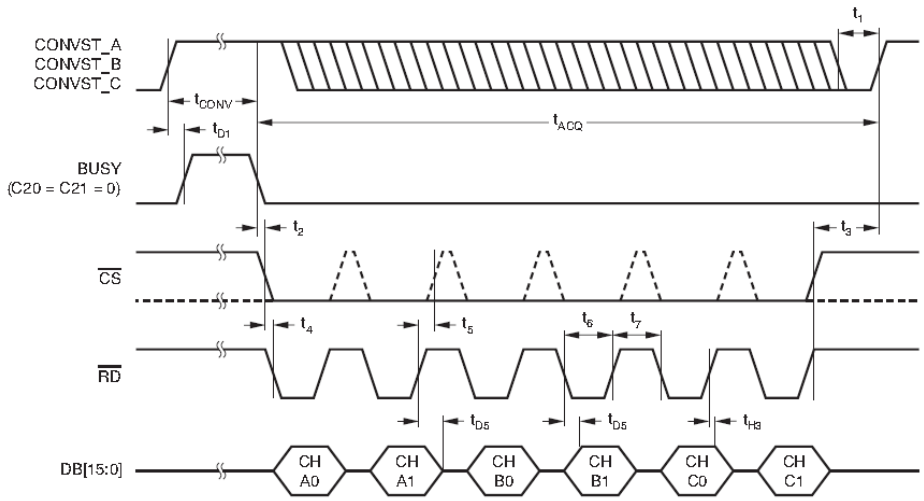

Figura 3.10: Diagrama de Tiempos para la lectura por protocolo paralelo

EL módulo implementado incluye una señal de verificación, llamada *Busy*, para conocer la disponibilidad del módulo para empezar una nueva adquisición, con la cual se puede concatenar una subsecuente accion que requiera los datos del ADC como por ejemplo la ejecución del módulo PID para generar la señal de control.

3.8. Comunicación

Como era necesario recibir datos desde fuera de la tarjeta se debió realizar una evaluación de los protocolos a implementar. La tarjeta *Spartan 3AN Starter Kit* posee puertos para comunicación Serial RS-232, PS-2, Ethernet y Jtag.

Para el desarrollo de este proyecto se dirigió la atención a la comunicación RS-232 y al JTAG[12].

- JTAG: Comprende un circuito implementado en la tarjeta de desarrollo por parte del fabricante, proveendo interfaz entre la FPGA y el computador mediante un puerto USB, logrando realizar operaciones de depuración de errores y programación de la FPGA. El software provisto por la empresa Xilinx cuenta con la documentación[13] y las herramientas para hacer uso en alto nivel de este protocolo de comunicación.

[12]Join Test Action Group, estandarizado mediante IEEE 1149.1

[13]http://www.xilinx.com/support/documentation/boards_and_kits/ug334.pdf

- Comunicación Serial RS-232: La tarjeta de desarrollo posee los circuitos para el acondicionamiento de las señales, debido a lo que se debió realizar un controlador UART para que manejara la comunicación. El módulo seleccionado fue el Uart?? provisto dentro de los IP[14] adquiridos con el sotware EDK. La decisión fue tomada evaluando el tiempo necesario en la implementación desde cero, el soporte provisto por el RTOS y Xilinx sobre el módulo, dando como vencedor al módulo uart?? provisto por Xilinx. La comunicación serial RS-232 será el protocolo de comunicación principal de este proyecto y dentro del sistema operativo se trabajará como entrada y salida estándar mediante el uso de las funciones *scanf* y *printf*.

3.9. Selección de RTOS

Como se dijo en el capítulo dos, los RTOS son herramientas de administración muy utiles dentro de los sistemas empotrados, por lo que se decidió utilizar una de estas piezas de software para que administre las operaciones de control sobre el robot MA2000. Para la selección de que RTOS se trabajó, se realizó una intensa búsqueda de los RTOS disponibles en el mercado, información condensada en el anexo B, en evaluaciones realizadas en conjunto por centros de investigación y empresas de CPUs[9, 10, 4], en las certificaciones del RTOS y en la disponibilidad del código fuente.

Característica	Valor
Máxima ROM necesaria	24kB
Mínima ROM necesaria	6kB
Número de servicios del Kernel	10 dierentes usando 80 llamadas API
Modelo de Multitarea	Basado en prioridades
Entidades de código de ejecución	Tareas y ISR[15]
Objetos Dinámicos	Estáticos y dinámicos
Movimiento de datos	Mensajes al buson y mensajes a colas
Semáforos	Si
Mutex	Si
Banderas de eventos	Si
Particiones de memoria	Si
Timers	Si

Tabla 3.8: Principales Características de μC/OS-II

Basados en las evaluaciones realizadas en el documento [9], se llegó a la conclusión de que el RTOS μC/OS-II es la mejor opción para trabajar, ya que presenta un excelente rendimiento en CPUs pequeñas y además provee el RTOS adaptado para gran variedad de arquitecturas, dentro de las que se encuentra el SoftCore MicroBlaze. En la tabla 3.8

[14]Intelectual Property

Nombre	Requiere RTOS
μC/TCP-IP	Si
μC/USB Host	No
μC/USB Device	No
μC/USB OTG	No
μC/FS	No
μC/CAN	No
μC/GUI	No
μC/Bluetooth	No

Tabla 3.9: Stacks de comunicaciones y Middleware de μC/OS-II

se muestran las características principales de μC/OS-II y en la tabla 3.9 se muestran los Stacks de comunicaciones y Middleware respectivos.

3.10. Drivers dentro del RTOS

Antes de entrar en materia es de vital importancia saber que el RTOS interactua con los periféricos mediante el BSP[16] de la tarjeta, el cual se encuentra en los archivos de texto llamados *bsp.h* y *bsp.c*. En el archivo *bsp.h* se definen diferentes constantes que representan los nombres de los periféricos dentro del sistema, y el archivo *bsp.c* provee diferentes funciones para inicializar y manejar los periféricos.

Dentro del trabajo realizado en el proyecto se crearon los drivers, consignados en la tabla 4.1, para manejar los leds, DipSwitch y LCD presentes en la tarjeta *Spartan 3AN Starter Kit*, mediante la integración de las funciones provistas por los módulos y los archivos *bsp.h* y *bsp.c*.

3.11. Sistema Final

Luego de haber diseñado o seleccionado cada uno de los componentes del sistema empotrado necesarios para satisfacer los requerimientos impuestos inicialmente, se procedió a integrarlos todos dentro de los dos prototipos finales.

Los componentes pertenecientes a la tarjeta *Spartan 3AN Starter Kit* fueron integrados con ayuda de la herramienta EDK[17], provista por la empresa Xilinx. El esquema final del sistema se muestran en la figura 3.11.

Los Elementos que no pertenecían a la tarjeta de desarrollo fueron agrupados dentro de una PCB, con ayuda del software Eagle,versión estudiantil provista por la empresa

[16]Board Support Package
[17]Embedded Development Kit

Driver	Descripción
BSP_LedsWr(INT8U leds)	Muestra el valor binario de la varible leds en los leds
BSP_DSRd()	Lee el valor binario representado por el DipSwitch
BSP_BtRd()	Lee el valor binario Obtenido de los Pulsadores de la Spartan3AN
BSP_LCD_PRINT_CHAR(INT8U pos, INT8U line, INT8U data)	Imprime Imprimir en LCD un caracter.
BSP_LCD_PRINT_REG(INT8U reg, INT32U data)	Imprime Todo un registro de 32bits a la LCD (4 Caracteres)
BSP_LCD_PRINT_HEX(INT8U reg, INT16U data)	Imprime un valor Numérico en representación BSD
BSP_LCD_CLEAN(void)	Limpia la pantala de la LCD
VSP_ADC_StartConv(void)	Inicia la Conversión de entradas analógicas
BSP_ADC_VerBusy(void)	Verifica la bandera de Ocupado del ADC
BSP_ADC_Conv(void)	Inicia una conversión y espera hasta su finalización.
BSP_ADC_GetData(INT8U Channel)	Obtiene el dato correspondiente a cada Canal
BSP_ADC_GetAndPrint(void)	Obtiene los datos de los 6 canales del ADC y los Imprime por el módulo Serial configurado
BSP_PID_StartCalc(INT32U Base_Adr, INT32U In_Uc, INT32U In_Y)	Inicial el Calculo del controlador PID para un determinado Canal
BSP_PID_VerBusy(INT32U Base_Adr)	Verifica la bandera de Ocupado de un canal de PID
BSP_PID_Calc(INT32U Base_Adr, INT32U In_Uc, INT32U In_Y)	Inicia el Calculo de un canal de controlador PID y espera a su finalización
BSP_PID_GetData(INT32U Base_Adr)	Obtiene el dato calculado en un canal del controlador PID
BSP_PID_Reset(INT32U Base_Adr)	Limpia los registros de un canal del controlador PID
BSP_PID_Config (INT32U Base_Adr, FP32 K, FP32 b, FP32 T, FP32 Ti, FP32 Td, FP32 N)	Permite configurar los parámetros del controlador PID
BSP_PWM_Set_DC(INT32U PWM_ADDR, INT16U DC_Divisor)	Permite Modificar el Valor de Ciclo Útil de un Módulo de PWM
BSP_PWM_Set_F(INT32U PWM_ADDR, INT16U F_Divisor)	Permite Modificar la Frecuencia de un Módulo de PWM
BSP_PWM_Set_CH(INT32U PWM_ADDR, INT8U Chanel)	Permite Cambiar de Canal de salida para invertir Dirección
BSP_PWM_Set_Delay(INT32U PWM_ADDR, INT8U Delay)	Permite modificar el retraso entre cambios de canal
BSP_PWM_Config(INT32U PWM_ADDR, INT8U DutyCycle, FP32 Frequency, INT8U Delay,INT8U Chanel)	Configuración General de Parámetros de los módulos PWM
BSP_PWM_Enable(INT32U PWM_ADDR)	Habilita un Módulo PWM
BSP_PWM_Disable(INT32U PWM_ADDR)	Dehabilita un módulo PWM
BSP_PWM_PID_IN(INT32U PWM_ADDR, INT32S DutyCycle, INT8U offset)	Recibe Datos del Módulo PID y permite seleccionar los 16 bits de interes.
BSP_PWM_Reset(INT32U PWM_ADDR)	Permite Reiniciar un Módulo PWM.

Tabla 3.10: Drivers Creados

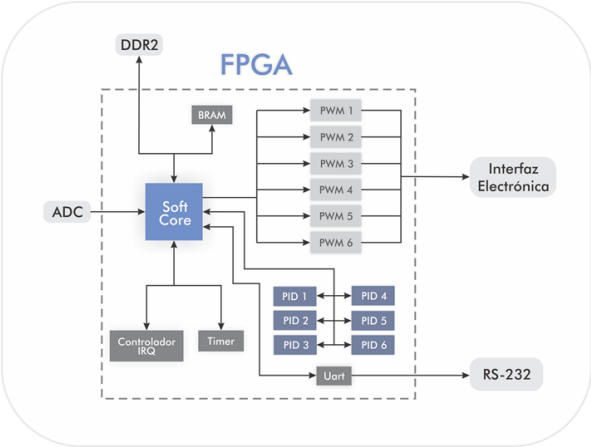

Figura 3.11: Sistema Dentro de la FPGA

CadSoft. La arquitectura de la PCB, puede ser apreciada en la figura 3.12 y diseño de la PCB se encuentra consignado en el Anexo D.

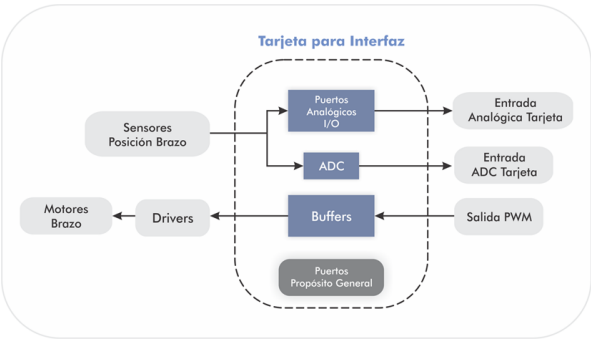

Figura 3.12: Interfaz Electrónica con MA2000

Capítulo 4

Validación del Sistema

Para la validación de los módulos PID y PWM se siguió un proceso de simulación, para luego ser integrados con Microblaze. En esta etapa tambíen se realizó la prueba de la tarjeta que provee la interfaz con el PUMA MA2000, ya que esta es la encargada de proveer la realimentación del sistema de control.

En la parte de software se escribió un algoritmo genérico adaptado al esquema de programación de un RTOS, donde se calculan los ángulos de los 6 grados de libertad, necesarios para lograr una posición y una orientación. Dentro del algoritmo se midieron los tiempos de ejecución de cada una de las tareas mediante funciones especializadas de μC/OS-II.

4.1. Módulo PID

4.1.1. Generación de patrón de Comparación

Utilizando una hoja de cálculo y el modelo discreto de un motor DC[1] de la ecuación 4.1, se simuló la respuesta de un sistema real ha ser controlado con un esquema de control PID implementado con Aritmética Distribuida, siguiendo las Ecuaciones 3.3. El utilizar esta hoja de cálculo permite obtener los valores teóricos de cada uno de los lazos del controlador, muestreo a muestreo, de manera rápida y precisa.

$$\frac{\theta(Z)}{V(Z)} = \frac{0{,}001 * Z + 0{,}001}{Z^2 - 1{,}9425 * Z + 0{,}9425} \tag{4.1}$$

Los cálculos necesarios en el algoritmo de DA se restringieron en rango y presición decimal, de manera que se contara con el efecto propio de tener números representados en punto fijo con número limitado de bits, tal y como se implementa en la FPGA.

[1]Disponible en http://www.engin.umich.edu/group/ctm/examples/motor2/motor.html

Para la implementación del controlador PID se utilizarón los siguientes valores en las constantes de control, considerarondo ocho bits para la representación decimal en representaciçon de punto fijo:

- $K = 27$
- $Ti = 0,6$
- $Td = 0,005$
- $b = 1$
- $N = 10000$
- $T = 0{,}001$

Los valores obtenidos con la Hoja de Calculo para los primeros 30 periodos de muestreo se muestran en el Anexo C.1, donde n es el número de muestra, Uc corresponde al *Setpoint*, Y sería el valor leído mediante el valor digitalizado de Y_Real, el cual a su vez corresponde a la salida esperada según el modelado discreto de la ecuación 4.1, P es el valor del lazo porporcional, I es el valor del lazo integral, D es el valor del lazo proporcional, PID es el valor total del controlador y PID_D es valor de PID en representación en punto fijo con 8 bits de precisión.

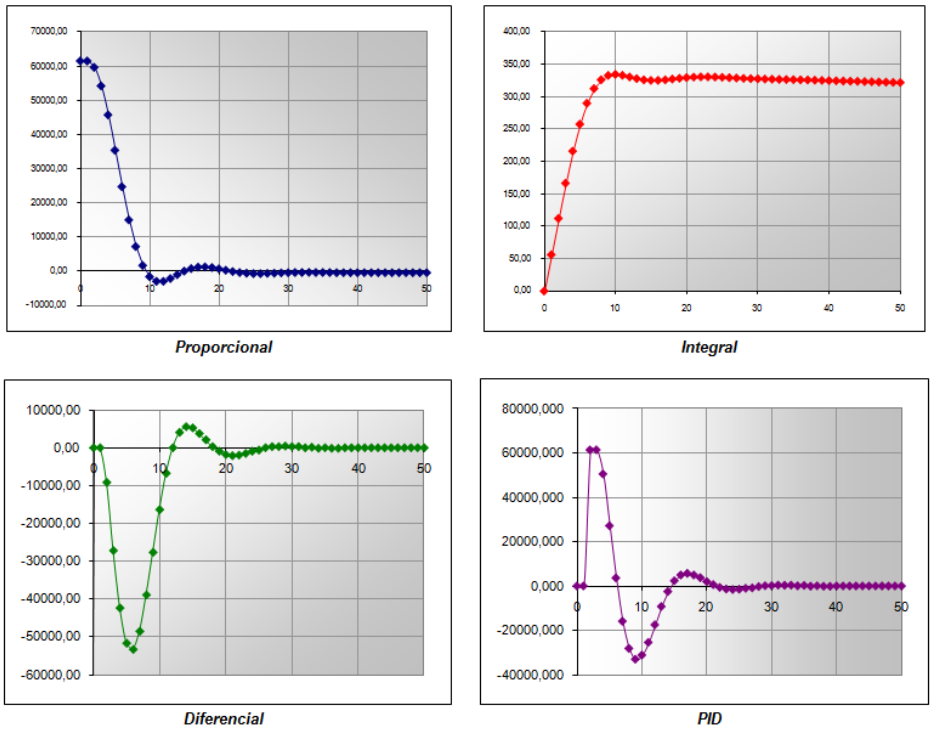

Figura 4.1: Respuesta de los Lazos de Control Proporcional, Integral, Diferencial y la salida completa del controlador PID

El patrón generado con la hoja de cálculo permite generar además gráficas del comportamiento esperado de los diferentes lazos del controlador y de la respuesta del sistema. Para nuestro caso de prueba se obtienen la figuras 4.1 y 4.2

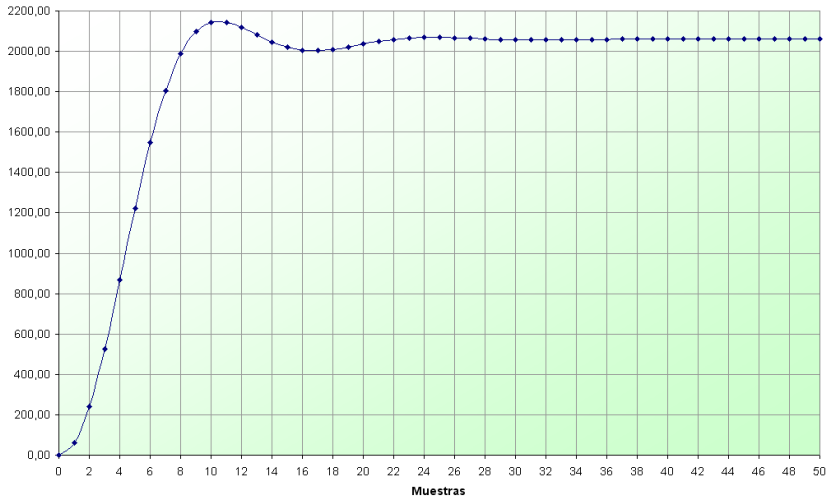

Figura 4.2: Respuesta Simulada del Sistema de Prueba

4.1.2. Simulación de controlador PID implementado con DA

Utilizando Xilinx ISE Design Suite se procedió a generar un archivo de simulación para el módulo de control PID, simulando la realimentación, la señal *Start* que permite que se ejecute el algoritmo (Como se puede observar en el diagrama de la Figura 3.6), la señal de Reinicio y por supuesto el reloj de referencia. El código del archivo de simulación generado se puede consultar en el Anexo C.2.

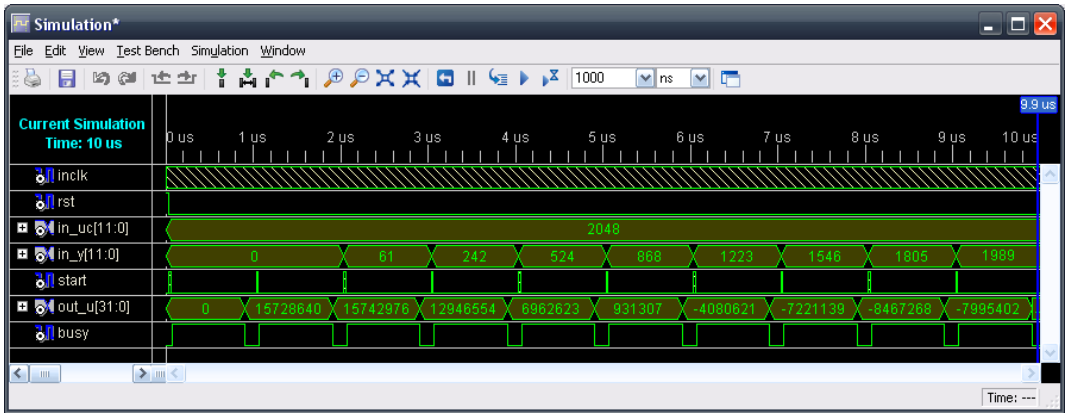

Figura 4.3: Simulación para los primeros nueve periodos de muestreo

Como resultado de la simulación realizada se puede verificar el correcto funcionamiento del modulo PID implementado con Aritmética Distribuida al comparar los valores de salida del módulo con la columna PID_D de la tabla presente en el Anexo C.1. La figura 4.3 muestra los resultados de simulación para los primeros nueve períodos de muestreo. Se debe tener en cuenta que aunque la señal *Start* en simulación no se encuenta cumpliendo el período de muestreo especificado por el patrón de comparación, lo que se busca verificar en esta prueba es que se efectuen de manera correcta los cálculos ante un flanco de subida de *Start* y que se genera la señal de realimentación mientras se realiza el cálculo.

4.1.3. Prueba de Implementación de controlador PID

Al igual que con el módulo de adquisición de datos desde el ADC, se creó un proyecto de Microblaze utilizando la herramienta Embedded Development Kit de XIlinx, incluyendo el módulo del controlador PID así como aquellos de visualización como el LCD y la comunicación Serial. Utilizando nuevamente las funciones disponibles en el BSP y las capacidades multitarea de uC-OSII se implementó una tarea encargada de simular el comportamiento de un motor, al mismo tiempo que realizaba control sobre este. Los valores de salida del módulo PID fueron enviados a traves del puerto serial, junto con el valor actual que se esperaría de la planta simulada, de manera que esta tabla se contrastara contra el patrón obtenido anteriormente con la hoja de cálculo. Los resultados fueron satisfactorios al seguir exactamente el patrón. El código utilizado se presenta a continuación:

```
INT32S PID[3]={0,0,0};
INT32U Y;
double Y_Real[3]={0,0,0};
int i=0;
printf("----------BEGIN----------\n");
while (i<30){
      //Muestreo (Y[n]=> "Y atrasada n muestras")
     Y=(INT32U)(Y_Real[1]);
     //Calculo Valor de Controlador
     PID_Calc((INT32U)(2048), (INT32U)(Y));
     PID[0]=PID_GetData();
     //Actuador
     //Respuesta de la Planta
     Y_Real[0]=(0.001*(PID[1]+PID[2])/256
                  +1.9425*(Y_Real[1])-0.9425*(Y_Real[2]));
     //Tiempo entre acciones de Control
     i++;
     Y_Real[2]=Y_Real[1];
     Y_Real[1]=Y_Real[0];
     PID[2]=PID[1];
     PID[1]=PID[0];
     printf("|n=%i\t|Y=%i\t|PID=%i\t|Y_Real=%i\t|\n",i,Y,
            PID[0],(INT32U)(Y_Real[0]));

}
```

4.2. Módulo PWM

4.2.1. Simulación del módulo PWM

El proceso de simulación se realizo con la herramienta ISim provista por la empresa Xilinx. En esta etapa se evaluó que el módulo produjera cada una de las señales en el tiempo especificado, así como que al cambiar de canal de salida se produjera la espera programada por el usuario. El código en lenguaje VHDL realizado para la prueba de este módulo puede ser consultado en la el anexo C.3.

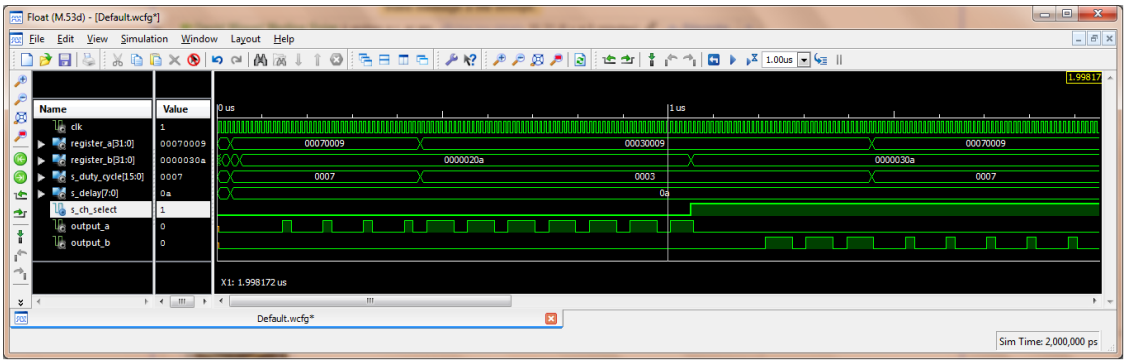

Figura 4.4: Simulación PWM

4.2.2. Prueba Física a módulo PWM

Con el objetivo de probar la funcionalidad final del módulo PWM que consiste en el manejo de los drivers de potencia de cada uno de los motores correspondientes a los seis grados de llibretad del brazo robot MA2000, se realizó una prueba que incluyera una etapa de potencia similar.

Para esto se utilizó una tarjeta de interfaz de potencia provista por el Grupo ERA, la cual dispone de dos Puentes de Potencia Duales de referencia L298 de ST Microelectronics. Esta tarjeta fue energizada en su etapa lógica con 5VDC provenientes de la tarjeta auxiliar y con 11,5VDC de una fuente externa para la etapa de potencia. Se adapto un cable de conexión al conector de 26 pines disponible en la tarjeta auxiliar con el fin de suplir las señales de PWM y de habilitación de los L298.

El módulo PWM se configuró mediante las funciones disponibles en el BSP de la tarjeta para tener una frecuencia base de 1kHz. La polaridad del PWM se manejó directamente con uno de los interruptores disponibles en la Spartan-3AN, y con los otros interruptores se seleccionó el porcentaje de ciclo útil deseado. Estos porcentajes fueron variados de manera que se realizara un barrido por todo el rango dinámico de las señales de PWM.

A la etapa de potencia se le acopló en principio una carga netamente resistiva, con la cual se realizarón pruebas de variación de ciclo útil a la señal de PWM. Posteriormente se utilizaron un motor de 24VDC y 3400 RPM en cada uno de los dos canales disponibles

como la carga de la etapa de potencia. Con esta segunda carga se realizaron pruebas de cambio de sentido de giro y de aislamiento de las señales PWM digitales y la etapa de potencia. Una imagen del montaje con los motores puede verse en la figura 4.5

Figura 4.5: Montaje para Prueba Física de PWM con etapa de Potencia

Un Ejemplo del Código en lenguaje C Utilizado para esta parte puede se muestra a continuación:

```
FP32 dc;
INT32U dips_val;
INT32U opc, activo, ch;
INT32U div_f,div_dc,dc_hight,dc_low;
INT32U f_clk,f_out,f_bus;
f_bus=62500000;
f_clk=1000000;
f_out=1000;
div_f=(f_clk/f_out);

XGpio Dips;
XGpio_Initialize(&Dips, XPAR_DIPS_4BIT_DEVICE_ID);

PWM_Set_F(BSP_PWM0_ADDR,(INT16U)(div_f));
PWM_Set_Delay(BSP_PWM0_ADDR, (INT8U)(50));
PWM_Enable(BSP_PWM0_ADDR);
```

```
    PWM_Set_F(BSP_PWM1_ADDR,(INT16U)(div_f));
    PWM_Set_Delay(BSP_PWM1_ADDR, (INT8U)(50));
    PWM_Enable(BSP_PWM1_ADDR);

    while (1){

    dips_val=XGpio_DiscreteRead(&Dips, 1);
    activo = (dips_val & (INT32U)(0x1));
    ch = (dips_val & (INT32U)(0x2)) >> 1;
    opc = (dips_val & (INT32U)(0xC)) >> 2;

    if( activo == 1 ){
        if(ch == 1){
            PWM_Set_CH(BSP_PWM1_ADDR, (INT8U)(1));
            PWM_Set_CH(BSP_PWM0_ADDR, (INT8U)(1));
        }
        else{
            PWM_Set_CH(BSP_PWM1_ADDR, (INT8U)(0));
            PWM_Set_CH(BSP_PWM0_ADDR, (INT8U)(0));
        }
        switch(opc){
            case 0:  dc_hight=25; break;
            case 1:  dc_hight=50; break;
            case 2:  dc_hight=75; break;
            default: dc_hight=100;
        }
        dc_low=100-dc_hight;
        dc=(float)(dc_low);
        dc=(f_out*100)/dc;
        div_dc=(u16)(f_clk/dc);
        PWM_Set_DC(BSP_PWM1_ADDR, (INT16U)(div_dc));
        PWM_Set_DC(BSP_PWM0_ADDR, (INT16U)(div_dc));
    }
    else{
        PWM_Disable(BSP_PWM1_ADDR);
        PWM_Reset(BSP_PWM1_ADDR);
        PWM_Disable(BSP_PWM0_ADDR);
        PWM_Reset(BSP_PWM0_ADDR);
    }
    OSTimeDlyHMSM(0,0,0,100);
     }
}
```

4.3. Módulo ADC

4.3.1. Simulación a periférico para ADC externo

Para asegurar en primera instancia la correcta implementación del hardware dedicado a la adquisición de datos del ADC, se realizó una simulación básica utilizando Xilinx ISE Design Suite, donde se le proporcionan al módulo los valores que provendrían del ADS8558 en la implementación real, en los instantes en que este los solicita, y se verifica que se este cumpliendo con el diagrama de tiempo de la Figura 3.10.

Una imagen de la simulación realizada se presenta en la Figura 4.6, donde se observa que se realizan las seis adquisiciones con los tiempos requeridos.

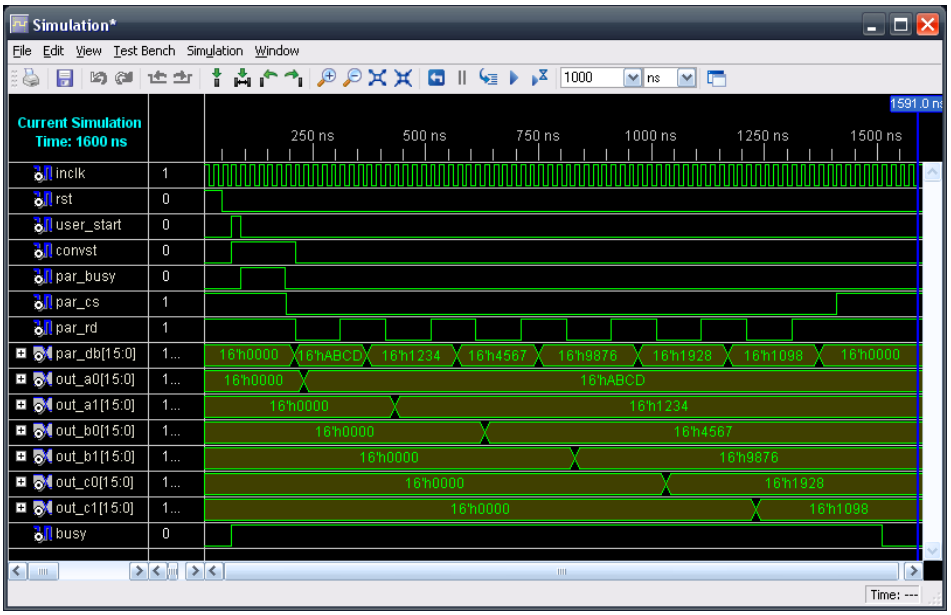

Figura 4.6: Simulación Protocolo Paralelo de adquisición de datos desde el ADS8558

4.3.2. Prueba Física a ADC externo

Posteriormente a la simulación se procedió a la prueba real del modulo ADC. Para esta prueba se creó un proyecto de Microblaze conteniendo el periférico del ADC implementado en hardware, al igual que el periférico para la LCD. Utilizando las funciones implementadas en el BSP como se muestra en la siguiente sección de código:

```
BSP_LCD_CLEAN();
BSP_LCD_PRINT_CHAR(0,1,0x07);
BSP_LCD_PRINT_CHAR(1,1,0x06);
BSP_LCD_PRINT_CHAR(2,1,0x07);
printf("----------BEGIN----------\n");
```

```
while(1){
    BSP_ADC_GetAndPrint();
    BSP_LCD_PRINT_HEX(2, ADC_GetData(1));
    BSP_LCD_PRINT_HEX(3, ADC_GetData(0));
    BSP_LCD_PRINT_HEX(4, ADC_GetData(3));
    BSP_LCD_PRINT_HEX(5, ADC_GetData(2));
    BSP_LCD_PRINT_HEX(6, ADC_GetData(5));
    BSP_LCD_PRINT_HEX(7, ADC_GetData(4));
}
```

Con esto se imprimia en Hexagesimal en la pantalla LCD cada uno de los valores leidos por el ADC, al igual que por el puerto serial. De esta manera se realizaron barridos por el rango dinámico del ADC utilizando fuentes de tensión para cada uno de los canales, obteniendo un seguimiento de muy alta precisión. Se evidenció sin embargo que a pesar de los seguidores de tensión y los filtros pasivos implementados en hardware en la tarjeta auxiliar se daba presencia de ruido en la lectura afectando principalmente los últimos 3 bits. Sin embargo lo anterior no supone un gran problema pues al ser los bits menos significativos no afectan la medida.

4.4. Sistema Completo

Como parte final del proceso de validación se presenta en la figura 4.7, que corresponde al sistema diseñado, dentro de la carcasa destinada para contener el sistema en su etapa final, además se muestra en la figura 4.8 el sistema de control provisto por la empresa TecQuiment LTD. a la hora de la compra del sistema.

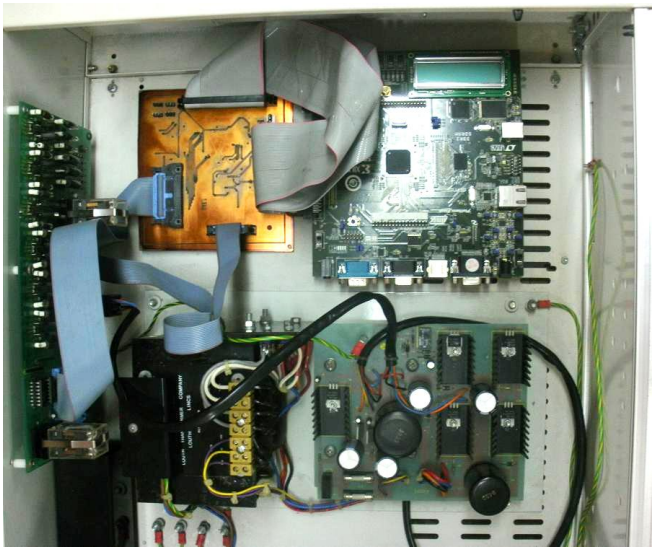

Figura 4.7: Sistema Diseñado

Dentro de los resultados obtenidos es importante destacar el consumo de recursos hardware dentro de la FPGA utilizada, por parte de cada uno de los módulos que conforman el sistema total.

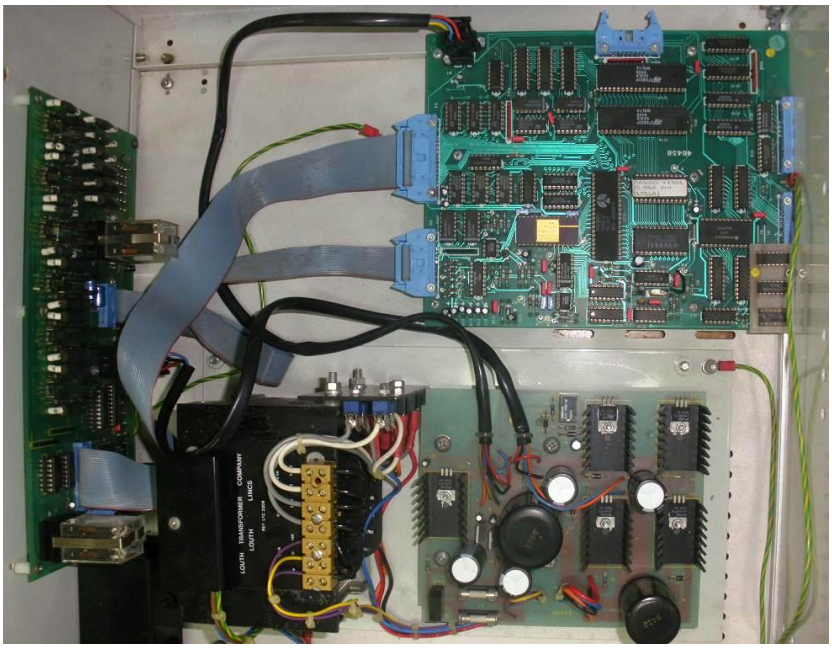

Figura 4.8: Sistema Provisto por TecQuiment LTD.

Módulo	Slice F/F	%	4 input LUT	%	Slices	%
ADC	109	0	49	0	70	1
LCD	530	4	71	0	277	4
PWM	100	0	34	0	52	0
PID	676	5	297	2	364	6
Microblaze	3.796	32	4.499	38	3.499	59

Tabla 4.1: Drivers Creados

4.5. Interfaz Electónica

A pesar de utilizar la tarjeta de desarrollo de la Spartan 3-AN como base de la plataforma del presente proyecto, se hace necesario la adecuación de las señales de entrada y salida a la FPGA, representadas principalmente por las realimentaciones de posición mediante los potenciometros y las señales PWM de control, así como asegurar la compatibilidad con algunos de los conectores manejados en la tarjeta de control de la plataforma original. Estas adecuaciones se aseguran mediante la adaptación de una tarjeta auxiliar a la tarjeta de desarrollo, la cual se interconecta mediante el Conector auxiliar de 100 pines de la marca HiRose, serie FX2, presente en esta. La tarjeta auxiliar sirve igualmente para la implementación del ADS8558 y su interconexión con la FPGA y las señales de entrada de los potenciometros en cada grado de libertad. Las diferentes vistas de la tarjeta, se pueden observar en las figuras 4.9 y 4.10.

El Esquematico y el Diseño PCB de la tarjeta auxiliar se realizaron con el software especializado EAGLE Layout Editor 4.16 en su versión estudiantil gratuita. El Esquemático

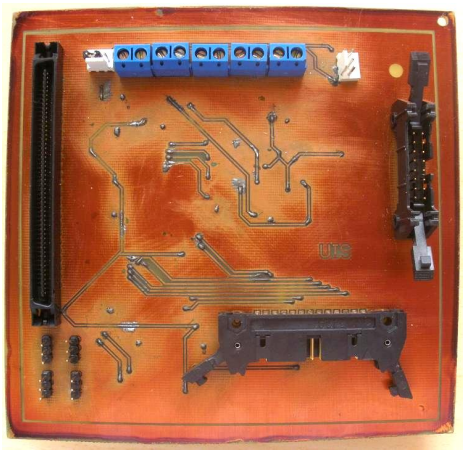

Figura 4.9: Vista Superior Interfaz

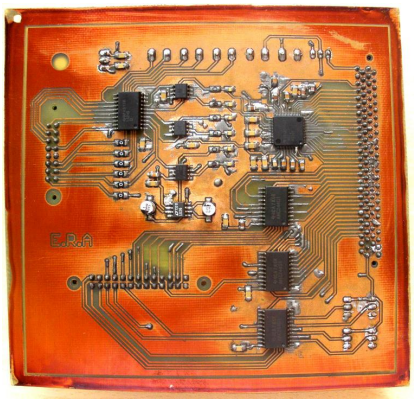

Figura 4.10: Vista Inferior Interfaz

y el Diseño PCB se presenta en las Figura 3.12 y en el apéndice D.1

4.6. Prueba Software

Como se nombra en [5] las ecuaciones para el cálculo de las variables articulares de un ROBOT de seis grados de libertad son:

$$
\begin{aligned}
q_1 &= tan^{-1}\left(\frac{p_y}{p_x}\right) \\
q_2 &= tan^{-1}\left(\frac{\sqrt{p_x^2+p_y^2}}{l_1-p_z}\right) \\
q_3 &= C_2(p_z-l_1)-S_2\sqrt{p_x^2+p_y^2} \\
q_4 &= sen^{-1}\left(\frac{r_{23}}{r_{33}}\right) \\
q_5 &= cos^{-1}\left(r_{33}\right) \\
q_6 &= sen^{-1}\left(-\frac{r_{32}}{r_{31}}\right)
\end{aligned}
\tag{4.2}
$$

donde q_i corresponde al ángulo de la articulación i-esima y r_{23}, r_{31}, r_{32} y r_{33} estan dadas dadas por:

$$
\begin{aligned}
r_{23} &= -S_1a_x + C_1a_y \\
r_{31} &= -C_1S_2n_x - S_1S_2n_y + C_2n_z \\
r_{32} &= -C_1S_2o_x - S_1S_2o_y + C_2o_z \\
r_{33} &= -C_1S_2a_x - S_1S_2a_y + C_2a_z
\end{aligned}
\tag{4.3}
$$

Teniendo en cuenta que n_x, n_y, n_z, o_x, o_y, o_z, a_x, a_y y a_z son constantes reales. Posteriormente

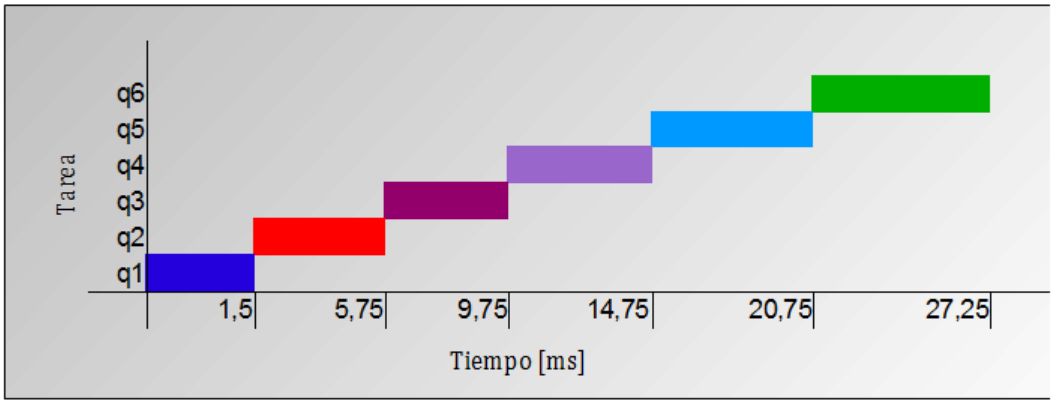

Figura 4.11: Tiempo de Ejecución de Tareas

a tener la descripción de las ecuaciones que se implementarian, se procedió a la programación dentro del RTOS, para la cual se siguió el esquema de distribución de tareas y prioridades

mostrado en la figura 4.11. Este esquema es una de las tantas formas en que se pueden designar y ejecutar las tareas. El objetivo del planeamiento planteado era tratar de evaluar una planeación de tareas y asignación de prioridades que produjera mayor carga a la CPU y obtener un dato de referencia para el peor de los casos.

Capítulo 5

Resultados

5.1. Conclusiones

- Se mostraron los pasos a tener en cuenta para la implementación de un Sistema Operativo de Tiempo Real en un sistema empotrado, basado en tecnología FPGA.
- La alta portabilidad y flexibilidad de μC/OS-II y los pocos requerimientos hardware y software de este lo hacen ideal para aplicaciones utilizando SoftCores.
- Se diseño una tarjeta que cumple con los requerimientos para ser utilizada en el control del sistema para el que fe planteada, brindando soporte a las conexiones estándar presentes en la tarjeta a reemplazar.
- Se dió soporte a la adecuación e interpretación de las señales de realimentación, así como de las salidas de control mediante PWM.
- Se validaron cada una de los periféricos desarrollados en hardware dentro de la FPGA, asi como del soporte software para el control de estos.
- Se validó el uso del RTOS μC/OS-II dentro del SoftCore Microblaze, implementado en la FPGA presente en la tarjeta de desarrollo Spartan-3AN, de manera que cumpla con los requerimientos de procesamiento, administración y control de las múltiples tareas que involucra el sistema.
- El sistema modular diseñado es capaz de suplir las necesidades de procesamiento para realizar tareas de control paralelo de 6 grados de libertad del robot MA2000, con implementación de una cinemática desacoplada.
- Se diseñó un sistema modular, que permite la modificación del número de grados de libertad a controlar.

5.2. Limitaciones

- La tarea de estabilidad del sistema se le encarga al usuario, por lo cual ante un mal diseño del controlador, se pueden presentar casos de *WindUp* o desborde en el registro

de salida del controlador.

- Actualmente el Módulo PID se encuentra configurado para una precisión decimal de hasta 8 bits. Si se requiere mayor precisión es imperativo reconfigurar el módulo y por tanto la estructua interna de la plataforma.
- La estructura interna de μC/OS-II y la frecuencia base trabajada para MicroBlaze limitan el tiempo mínimo de procesamiento para un algoritmo de cálculo de trayectorias a 30ms
- La resolución en frecuencia del módulo de PWM esta ligado directamente al número de bits asignado al contador interno de referencia, así como a la velocidad de reloj del módulo.
- El utilizar el ADS8558 en configuración paralelo implica un mayor gasto de pines de propósito general de la FPGA. Si se configura en transmisión serial sería necesaria la implementación del protocolo de comunicación, pero se podría disponer de mas conexiones a lógica o circuitería externa.
- La frecuencia máxima de operación para el módulo de cálculo PID con la configuración actual es de 35,275MHz.
- La frecuencia máxima de operación para el módulo de generación PWM es de 126,358MHz.
- La plataforma de desarrollo SPARTAN-3AN desarrollado alrededor del Integrado *xc3s700anfgg484*, no provee los recursos lógicos suficientes para albergar el sistema diseñado controlando los 6 grados de libertad del Robot PUMA MA2000.
- Estando fuera de los alcances de este proyecto, no se presenta la implementación Software final del control sobre el Brazo Robot MA2000.

5.3. Recomendaciones

- Optimizar el diseño Hardware y Software del módulo PID para contemplar filtro AntiWindUp, Limitadores o posibles aproximaciones por ecuaciones discretas de mayor grado. Para lo anterior se puede aprovechar la reconfigurabilidad de los parámetros de Salida de las LUT actuales, o implementar algún otro tipo de lógica en DA como la presentada en [7].
- En futuros trabajos se recomienda indagar en la Aritmética Distribuida y sus posibles aplicaciones utilizando formato estándar en coma flotante, o en punto fijo pero representación fraccionaria.
- Para mejorar el uso de recursos lógicos dentro del sistema, se recomienda multilexar en tiempo un único controlador PID. Esta multiplexación sería dependiente de la aplicación final, por lo que se posterga para el posterior diseño software a realizarse sobre la plataforma.

- Analizar diferentes protocolos para la adquisición de las señales analógicas mediante ADC, en búsqueda de mantener optima la taza de muestreo máxima y los requisitos hardware.
- Se recomienda la interconexión de la tarjeta de interfaz electrónica de este proyecto, con diferentes plataformas de simulación, o mediante tarjetas de adquisición, de manera que se puedan aplicar conceptos básicos de control y de sistemas embebidos, así como para realizar un análisis más detallado de las prestaciones de esta.
- Adecuar la plataforma MA2000 en su estructura Hardware para realizar pruebas posteriores a la finalización del presente proyecto, donde se puedan seguir rutinas básicas de posicionamiento.
- Para la aplicación final de la presente tarjeta en un sistema completo dedicado a controlar el brazo robot MA2000, se recomienda habilitar conectividad tipo Ethernet.
- Utilizar una plataforma de desarrollo de mayor capacidad de bloques lógicos para albergar el control de los seis grados de libertad que posee el brazo robot MA2000. Se recomienda el trabajo con la misma familia de FPGA Spartan-3AN pero con un desarrollo alrededor del chip *XC3S1400AN*[1]
- En una futura implementación final de todo el sistema, se recomienda rediseñar de control, basándose en alguna plataforma de desarrollo como la Spartan-3AN, eliminando aquellos elementos que no se requieran e insertando en una misma PCB los elementos que actualmente son externos.

[1]Información de la familia Spartan-3AN disponible en el documento "DS557 - Spartan-3AN FPGA Family Data Sheet"de Xilinx

Bibliografía

[1] TecQuipment LTD. *Manual de Funcionamiento robot MA2000*, 1995.

[2] Wayne Wolf. *High Performance Embedded Computing*. Morgan Kauffman, 2006.

[3] Fred Eady Creed Huddleston Lewin Edwards David J. Katz Rick Gentile Ken Arnold Kamal Hyder Jack Ganssle, Tammy Noergaard and Bob Perrin. *Embedded Hardware: Know it all*. Elsevier Inc., 2008.

[4] S Tan and B Tran Nguyen. Survey and performance evaluation of real-time operating systems (rtos) for small microcontrollers. *Micro, IEEE*, 2009.

[5] A. Barrientos. *Fundamentos De Robotica*. Mcgraw-hill, 2007.

[6] M.S.M. Siddiqui, A.H. Sajid, and D.G. Chougule. Fpga based efficient implementation of pid control algorithm. In *Control, Automation, Communication and Energy Conservation, 2009. INCACEC 2009. 2009 International Conference on*, pages 1 –5, 2009.

[7] Y.F. Chan, M. Moallem, and W. Wang. Efficient implementation of pid control algorithm using fpga technology. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 5, pages 4885 – 4890 Vol.5, 2004.

[8] Inc. Xilinx. The role of distributed arithmetic in fpga-based signal processing. page 15, 1997.

[9] A.K. Ben Salem, S. Ben Othman, and S. Ben Saoud. Hard and soft-core implementation of embedded control application using rtos. In *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, 30 2008.

[10] Su-Lim Tan and Tran Nguyen Bao Anh. Real-time operating system (rtos) for small (16-bit) microcontroller. In *Consumer Electronics, 2009. ISCE '09. IEEE 13th International Symposium on*, pages 1007 –1011, May 2009.

[11] Gina R. Smith. *FPGA101 - Everything You need to know to get started*. Elsevier Inc., 2010.

[12] Andrew G. Schmidt Ron Sass. *Embedded Systems Design with Platform FPGAs*. Elsevier Inc., 2010.

[13] Anthony Massa Michael Barr. *Programming Embedded Systems , publisher =.*

[14] Balch M. *Complete Digital Design.* Mcgraw-hill, 2003.

Apéndices

Apéndices A

SoftCores disponibles en el mercado

En la actualidad el interés creciente por este tipo de plataformas ha llevado al diseño y desarrollo de múltiples SoftCore por diferentes empresas, universidades y desarrolladores particulares. Cada uno de estos SoftCores posee diferente arquitectura, requisitos lógicos, reconfigurabilidad, periféricos y soporte, cada uno, dependiendo de la cantidad de desarrolladores y del objetivo final del procesador. A continuación se muestra una tabla con las CPUs más relevantes hasta el día de hoy y sus principales características[1].

Núcleo de la CPU	Arquitectura	Bits	Licencia	Profundidad de Pipeline	Ciclos por instrucción	MMU	MUL	FPU	Área
S1 Core	SPARC-v9	64	GPL	6	1	Si	Si	Si	37000-60000
LEON3	SPARC-v8	32	GPL	7	1	Si	Si	Si	3500
LEON2	SPARC-v8	32	LGPL	5	1	Si	Si	Ext	5000
OpenRISC 1200	OpenRISC 1000	32	LGPL	5	1	Si	Si	Ext	6000
MicroBlaze	MicroBlaze	32	Propietaria	3-5	1	Opt	Opt	Opt	1324
aeMB	MicroBlaze	32	LGPL	3	1	No	Opt	-	2536
OpenFire	MicroBlaze	32	MIT	3	1	No	Opt	-	1928
Nios II/f	Nios II	32	Propietaria	6	1	Si	Si	Opt	1800
Nios II/s	Nios II	32	Propietaria	5	1	No	Si	Opt	1770
Nios II/e	Nios II	32	Propietaria	No	6	No	No	Opt	390
LatticeMicro32	LaticeMicro32	32	Open Source	6	1	No	Opt	No	1984
Cortex-M1	ARMv6	32	Propietaria	3	1	No	Si	No	2600
DSPuva16	DSPuva16	16	Open Source	No	4	No	Si	No	510
PicoBlaze	PicoBlaze	8	Propietaria	No	2	No	No	No	192
PicoBlaze	PicoBlaze	8	BSD	No	2	No	No	No	204
LaticeMicro8	LaticeMicro8	8	Open Source	No	2	No	No	No	200

[1] http://www.1-core.com/library/digital/soft-cpu-cores/#table_ref5

Apéndices B

RTOS disponibles en el mercado

Tabla B.1: RTOS Disponibles en el Mercado

Nombre	Licencia	Modelo de código Fuente	Objetivo	Plataforma
ARTOS	Propietaria	Indefinido	Empotrado	ARM9
AVIX	Propietaria	Cerrado	Empotrado	PIC24F, PIC24H, dsPIC30F, dsPIC33F, PIC32MX
BeRTOS	GNU GPL	Código abierto	Empotrado	DSP56K, I196, IA32, ARM, AVR
ChibiOS/RT	GNU GPL	Código abierto	Empotrado	x86, ARM7, ARM Cortex-M3, PowerPC e200z, STM8, AVR, MSP430, Coldfire, H8S
CMX RTOS	Propietaria	Indefinido	Empotrado	IA32, ARM, AVR, H8, PIC, 8051
Contiki	BSD	Código abierto	Empotrado	MSP430, AVR
Deos	Propietaria	Cerrada	Seguridad Critica	x86, PowerPC, PowerQUICC
DSPnano RTOS	Open Source y Comercial	Código abierto	Empotrado	R8C, M16C, PIC24, dsPIC33
eCos	GNU GPL	Código abierto	Propósito General	ARM/XScale, CalmRISC, 68000/Coldfire, fr30, FR-V, H8, IA32, MIPS, MN10300, OpenRISC, PowerPC, SPARC, SuperH, V8xx
eCosPro	Propietaria	Código abierto	Propósito General	ARM/XScale, CalmRISC, 68000/Coldfire, fr30, FR-V, H8, IA32, MIPS, MN10300, NIOS2, OpenRISC, PowerPC, SPARC, SuperH, V8xx

Tabla B.1: RTOS Disponibles en el Mercado. Continuación.

Nombre	Licencia	Modelo de código Fuente	Objetivo	Plataforma
embOS	Propietaria	Cerrado	Empotrado	ARM,Nios II,Microblaze, X86, PIC18, AVR, PIC24, PIC32, Zilog, NEC, 8081, Renesas
Erika Enterprise	GNU GPL	Código abierto	Varios	ARM7, H8 (Hitachi), Nios2 (Altera), dsPIC33 (Microchip), ST10 (ST Microelectronics)/C167 (Infineon)
Femto OS	GNU GPLv3	Código abierto	Empotrado	AVR
FreeOSEK	GNU GPLv3	Código abierto	Empotrado	ARM7
FreeRTOS	GNU GPL	Código abierto	Empotrado	ARM, AVR, AVR32, Freescale ColdFire, HCS12, IA32, MicroBlaze, MSP430, PIC, Renesas H8/S, 8052, STM32
FunkOS	Sleepycat	Código abierto	Empotrado	AVR, MSP430, ARM Cortex-M3
Fusion RTOS	Libre	Cerrado	Proposito semi general	ARM, Blackfin, StarCore, DSP 56800E
HeartOS	Propietario	Cerrado	Cuidado Crítico	x86, PowerPC, PowerQUICC
Helium	Libre	Código Abierto	Empotrado	x86, HCS08 y AVR
INTEGRITY	Propietaria	Indefinido	Empotrado, alta seguridad	ARM, XScale, Blackfin, Freescale ColdFire, MIPS, PowerPC, x86
LynxOS	Propietaria	Código abierto	Empotrado	Motorola 68010, x86/IA-32, ARM, Freescale PowerPC, PowerPC 970, LEON3
MQX	Libre	Código abierto	Empotrado	CPUs de Freescale
MontaVista Linux	Propietaria	Cerrado	Empotrado	ARM, X86, CPUs Motorola, NEC, Panasonic.
Neutrino	Propietaria	Código abierto	Microkernel	ARM, MIPS, PPC, SH, x86, XScale

Tabla B.1: RTOS Disponibles en el Mercado. Continuación.

Nombre	Licencia	Modelo de código Fuente	Objetivo	Plataforma
Nucleus OS	Propietaria	Código abierto	Empotrado	AMD Au1100, ARM, Atmel AT91 series, Atmel Nios II, Freescale iMX, Freescale MCF, Freescale MPC, Marvell PXA series, MTI, NEC uPD6111x, Sharp LH7 series, ST, TI OMAP, TI TMS320 series, Xilinx Microblaze
On Time RTOS-32	Propietaria	Código abierto	Microkernel	32/64-bit x86
Open AT OS	Propietaria	Cerrado	Dispositivos GSM Empotrados	ARM
OSE	Propietaria	Cerrado	Propósito general	ARM, PowerPC, MIPS, IXP2400, TI OMAP
OS-9	Propietaria	Cerrado	Uso industrial	ARM/strongARM, MIPS, PowerPC, SuperH, x86/Pentium, XSCALE, Motorola 6809, Motorola 68000-series
Phoenix-RTOS	GNU GPL	Código abierto	Empotrado	ARM7, X86, PowerPC
PikeOS	Propietaria	Disponible a la venta	Cuidado critico	PPC, x86, ARM, MIPS, SPARC/Leon, SuperH
Portos	Propietaria	Código abierto	Empotrado	DSP/BIOS
POK	BSD	Código abierto	Empotrado	x86, PowerPC, SPARC
Prex	BSD	Código abierto	Microkernell	ARM, IA32
QNX	Mixta	Disponible para uso personal	Propósito General	IA32, MIPS, PowerPC, SH-4, ARM, StrongARM, XScale
Q-Kernell	Propietario	Disponible a la venta	Empotrado	PIC-30, PIC-24, dsPIC, PIC32MX
RTAI	GNU GPL	Código Abierto	Empotrado	x86, ARM
RTEMS	GNU GPL	Código abierto	Empotrado	PIC-30, ARM, Blackfin, ColdFire, TI C3x/C4x, H8/300, x86, 68k, MIPS, Nios II, PowerPC, SuperH, SPARC, ERC32, LEON, Mongoose-V

Tabla B.1: RTOS Disponibles en el Mercado. Continuación.

Nombre	Licencia	Modelo de código Fuente	Objetivo	Plataforma
rt-kernel	Propietario	Disponible a la venta	Empotrado	ARM, Cortex-M3, Blackfin, PowerPC, Windows(simulation)
RTLinux	GNU GPL	Código abierto	Propósito general	X86,ARM
RTXC Quadros	Propietaria	Código abierto	Empotrado	ARM:Atmel /Freescale/NXP/ST/TI, Blackfin, Coldfire/68K, PowerPC, StarCore, TI/Luminary Stellaris, TI OMAP, XScale
SCIOPTA	Propietaria	Cerrado	Empotrado con alta seguridad	ARM, Cortex-M3, Cortex-M0, XScale, PowerPC, ColdFire, HCS12, M16C, MSP430
SDPOS	GNU LGPL	Código abierto	Empotrado	ARM, Cortex-M3, Blackfin, PIC18, PIC24, i386
Simple AVROS	Privativa	Código abierto	Empotrado	CPUs AVR
SymbianOS	Privativa	Código abierto	Dispositivos GSM empotrado	ARM
ThreadX	Privativa	Disponible a la venta	Empotrado	ARC, ARM/Thumb, AVR32, BlackFin, ColdFire/68K, H8/300H, Luminary Micro Stellaris, M-CORE, MicroBlaze, PIC24/dsPIC, PIC32, MIPS, V8xx, Nios II, PowerPC, SH, SHARC, StarCore, STM32, StrongARM, TMS320C54x, TMS320C6x, x86/x386, XScale, Xtensa/Diamond, ZSP
Trampoline Operating System (OSEK)	GNU LGPL	Código abierto	Empotrado	AVR, H8/300H, POSIX, NEC V850e, ARM7, Infineon C166, HCS12 y PowerPC
TNKernel	BSD	Código abierto	Empotrado	ARM, PIC24/dsPIC, HCS08

Tabla B.1: RTOS Disponibles en el Mercado. Continuación.

Nombre	Licencia	Modelo de código Fuente	Objetivo	Plataforma
uC/OS-II	Propietaria	Disponible bajo licencia	Empotrado	ARM7/9/11/Cortex M1/3, AVR, HC11/12/S12, Coldfire, Blackfin, Microblaze, NIOS, 8051, x86, Win32, H8S, M16C, M32C, MIPS, 68000, PIC24/dsPIC33/PIC32, MSP430, PowerPC, SH, StarCore, STM32
uTasker	Propietaria	Disponible para uso personal	Empotrado	Coldfire M522XX, AVR32, SAM7X, Luminary Micro, LPC2XXX, STR91X, NE64
u-velOSity	Propietaria	Cerrado	Microkernel	ARM
velOSity	Propietaria	Cerrado	Microkernel	Power Architecture, ARM/XScale, MIPS, x86/Pentium, ColdFire, Blackfin, OMAP, DaVinci
VRTX	Propietaria	Cerrado	Microkernel	ARM, MIPS, PowerPC
VxWorks	Propietaria	Cerrado	Empotrado	ARM, IA32, MIPS, PowerPC, SH-4, StrongARM, xScale
Windows CE	Propietaria	Microsoft shared source	Empotrado	x86, MIPS, ARM, SuperH
Xenomai	GNU GPL	Código abierto	Empotrado	x86, x86-64, PowerPC, ARM, Analog Devices Blackfin BF52x, BF53x, BF54x and BF56x
xPC Tarjet	Propietaria	Cerrado	Pruebas de tiempo real	x86

Apéndices C

Códigos de Pruebas

C.1. Patrón de Comparación PID

n	Uc	Y	P	I	D	PID	PID_D	Y_{Real}
-1	0	0	0	0,00	0,00	0,000	0	0,00
0	2048	0	61440	0,00	0,00	61440,000	15728640	0,00
1	2048	0	61440	56,00	0,00	61496,000	15742976	61,44
2	2048	61	59610	112,00	-9149,52	50572,477	12946554	242,28
3	2048	242	54180	166,33	-27148,59	27197,746	6962623	524,80
4	2048	524	45720	215,71	-42297,80	3637,918	931307	868,84
5	2048	868	35400	257,39	-51597,31	-15939,926	-4080621	1223,93
6	2048	1223	24750	289,65	-53247,23	-28207,574	-7221139	1546,30
7	2048	1546	15060	312,21	-48447,48	-33075,266	-8467268	1805,99
8	2048	1805	7290	325,94	-38847,98	-31232,039	-7995402	1989,46
9	2048	1989	1770	332,58	-27598,56	-25495,980	-6526971	2098,08
10	2048	2098	-1500	334,20	-16349,15	-17514,953	-4483828	2143,72
11	2048	2143	-2850	332,83	-6749,65	-9266,820	-2372306	2143,73
12	2048	2143	-2850	330,23	0,00	-2519,770	-645061	2116,96
13	2048	2116	-2040	327,63	4049,79	2337,422	598380	2079,93
14	2048	2079	-930	325,77	5549,71	4945,484	1266044	2044,86
15	2048	2044	120	324,93	5249,73	5694,652	1457831	2019,08
16	2048	2019	870	325,04	3749,80	4944,840	1265879	2005,43
17	2048	2005	1290	325,83	2099,89	3715,719	951224	2003,20
18	2048	2003	1350	327,00	299,98	1976,988	506109	2009,76
19	2048	2009	1170	328,23	-899,95	598,281	153160	2021,64
20	2048	2021	810	329,30	-1799,91	-660,605	-169115	2035,41
21	2048	2035	390	330,04	-2099,89	-1379,852	-353242	2048,32
22	2048	2048	0	330,39	-1949,90	-1619,504	-414593	2058,45
23	2048	2058	-300	330,39	-1499,92	-1469,527	-376199	2065,00
24	2048	2064	-480	330,12	-899,95	-1049,832	-268757	2068,08
25	2048	2068	-600	329,68	-599,97	-870,285	-222793	2068,47
26	2048	2068	-600	329,14	0,00	-270,863	-69341	2066,91
27	2048	2066	-540	328,59	299,98	88,574	22675	2064,31
28	2048	2064	-480	328,10	299,98	148,082	37909	2061,67
29	2048	2061	-390	327,66	449,98	387,637	99235	2059,42
30	2048	2059	-330	327,30	299,98	297,289	76106	2057,83

C.2. Código Prueba PID implementado con DA

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_arith.ALL;

ENTITY Test_DA_PID IS
END Test_DA_PID;

ARCHITECTURE behavior OF Test_DA_PID IS

  COMPONENT DA_PID
  PORT(
    inclk : IN  std_logic;
    rst   : IN  std_logic;
    In_Uc : IN  std_logic_vector(11 downto 0);
    In_Y  : IN  std_logic_vector(11 downto 0);
    Out_U : OUT std_logic_vector(31 downto 0);
    P1, P2, P3, P4 : IN  std_logic_vector(19 downto 0);
    I1, I2, I3, I4 : IN  std_logic_vector(19 downto 0);
    I5, I6, I7, I8 : IN  std_logic_vector(19 downto 0);
    D1, D2, D3, D4 : IN  std_logic_vector(19 downto 0);
    D5, D6, D7, D8 : IN  std_logic_vector(19 downto 0);
    Start : IN  std_logic;
    Busy  : OUT std_logic);
  END COMPONENT;

  --Inputs
  signal inclk:std_logic := '0';
  signal rst  :std_logic := '0';
  signal In_Uc:std_logic_vector(11 downto 0):=(others=>'0');
  signal In_Y :std_logic_vector(11 downto 0):=(others=>'0');
  signal P1,P2:std_logic_vector(19 downto 0):=(others=>'0');
  signal P3,P4:std_logic_vector(19 downto 0):=(others=>'0');
  signal I1,I2:std_logic_vector(19 downto 0):=(others=>'0');
  signal I3,I4:std_logic_vector(19 downto 0):=(others=>'0');
  signal I5,I6:std_logic_vector(19 downto 0):=(others=>'0');
  signal I7,I8:std_logic_vector(19 downto 0):=(others=>'0');
  signal D1,D2:std_logic_vector(19 downto 0):=(others=>'0');
  signal D3,D4:std_logic_vector(19 downto 0):=(others=>'0');
  signal D5,D6:std_logic_vector(19 downto 0):=(others=>'0');
  signal D7,D8:std_logic_vector(19 downto 0):=(others=>'0');
  signal Start:std_logic := '0';
```

```
  --Outputs
  signal Out_U : std_logic_vector(31 downto 0);
  signal Busy  : std_logic;

  -- Clock period definitions
  constant inclk_period : time := 20ns;

BEGIN

  uut: DA_PID PORT MAP (
    inclk => inclk,
    rst   => rst,
    In_Uc => In_Uc,
    In_Y  => In_Y,
    Out_U => Out_U,
    P1 => P1, P2 => P2, P3 => P3, P4 => P4,
    I1 => I1, I2 => I2, I3 => I3, I4 => I4,
    I5 => I5, I6 => I6, I7 => I7, I8 => I8,
    D1 => D1, D2 => D2, D3 => D3, D4 => D4,
    D5 => D5, D6 => D6, D7 => D7, D8 => D8,
    Start => Start,
    Busy => Busy);

  -- Clock process definitions
  inclk_process :process
  begin
    inclk <= '0';
    wait for inclk_period/2;
    inclk <= '1';
    wait for inclk_period/2;
  end process;

  -- Stimulus process
  P1 <=x"00000"; P2 <=x"FE200"; P3 <=x"01E00"; P4 <=x"00000";
  I1 <=x"00000"; I2 <=x"FFFF9"; I3 <=x"00007"; I4 <=x"00000";
  I5 <=x"00100"; I6 <=x"000F9"; I7 <=x"00107"; I8 <=x"00100";
  D1 <=x"00000"; D2 <=x"095FE"; D3 <=x"F6A02"; D4 <=x"00000";
  D5 <=x"00000"; D6 <=x"095FE"; D7 <=x"F6A02"; D8 <=x"00000";

  stim_proc: process
  begin
    rst <= '1';
    In_Uc <= CONV_STD_LOGIC_VECTOR(2048,12);
    wait for 35ns;
    rst <= '0';
```

```
    for i in 0 to 10 loop
      case i is
        when  2 => In_Y <= CONV_STD_LOGIC_VECTOR(  61,12);
        when  3 => In_Y <= CONV_STD_LOGIC_VECTOR( 242,12);
        when  4 => In_Y <= CONV_STD_LOGIC_VECTOR( 524,12);
        when  5 => In_Y <= CONV_STD_LOGIC_VECTOR( 868,12);
        when  6 => In_Y <= CONV_STD_LOGIC_VECTOR(1223,12);
        when  7 => In_Y <= CONV_STD_LOGIC_VECTOR(1546,12);
        when  8 => In_Y <= CONV_STD_LOGIC_VECTOR(1805,12);
        when  9 => In_Y <= CONV_STD_LOGIC_VECTOR(1989,12);
        when 10 => In_Y <= CONV_STD_LOGIC_VECTOR(2098,12);
        when others => In_Y <= CONV_STD_LOGIC_VECTOR(0,12);
      end case;
      Start <= '1';
      WAIT FOR 20 ns;
      Start <= '0';
      WAIT FOR 980 ns;
    end loop;
    wait;
  end process;

END;
```

C.3. Código Prueba PWM con FSM

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY TB_PWM_PRU1 IS
END TB_PWM_PRU1;

ARCHITECTURE behavior OF TB_PWM_PRU1 IS

-- Component Declaration for the Unit Under Test (UUT)
COMPONENT PWM_DUAL_WITH_DELAY
PORT(
      REGISTER_A : IN  std_logic_vector(31 downto 0);
      REGISTER_B : IN  std_logic_vector(31 downto 0);
      CLK : IN  std_logic;
      OUTPUT_A : OUT  std_logic;
      OUTPUT_B : OUT  std_logic
     );
END COMPONENT;
--Inputs
signal REGISTER_A : std_logic_vector(31 downto 0) := (others => '0');
signal REGISTER_B : std_logic_vector(31 downto 0) := (others => '0');
signal CLK : std_logic := '0';
--Outputs
signal OUTPUT_A : std_logic;
signal OUTPUT_B : std_logic;
-- Clock period definitions
constant CLK_period : time := 10 ns;

BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: PWM_DUAL_WITH_DELAY PORT MAP (
       REGISTER_A => REGISTER_A,
       REGISTER_B => REGISTER_B,
       CLK => CLK,
       OUTPUT_A => OUTPUT_A,
       OUTPUT_B => OUTPUT_B
     );
-- Clock process definitions
CLK_process :process
begin
CLK <= '0';
wait for CLK_period/2;
```

```
CLK <= '1';
wait for CLK_period/2;
end process;
--USER DEFINITIONS
Registros_process: process
begin
 wait for 10 ns;
 --reset
 REGISTER_B <= X"00000400";
 wait for 2*CLK_period;
 --configuración
 REGISTER_A <= X"00070009";
 REGISTER_B <= X"0000000A";
 wait for 2*CLK_period;
 --start
 REGISTER_B <= X"0000020A";
 wait for 40*CLK_period;
 --cabio de ciclo de trabajo
 REGISTER_A <= X"00030009";
 wait for 60*CLK_period;
 --cambio de canal
 REGISTER_B <= X"0000030A";
 wait for 40*CLK_period;
 REGISTER_A <= X"00070009";
 wait;
end process;
-- Stimulus process
stim_proc: process
begin
  -- hold reset state for 100 ns.
   wait for 10 ns;
   wait for CLK_period*10;
   wait;
end process;

END;
```

Apéndices D

Tarjeta para Interfaz Electrónica

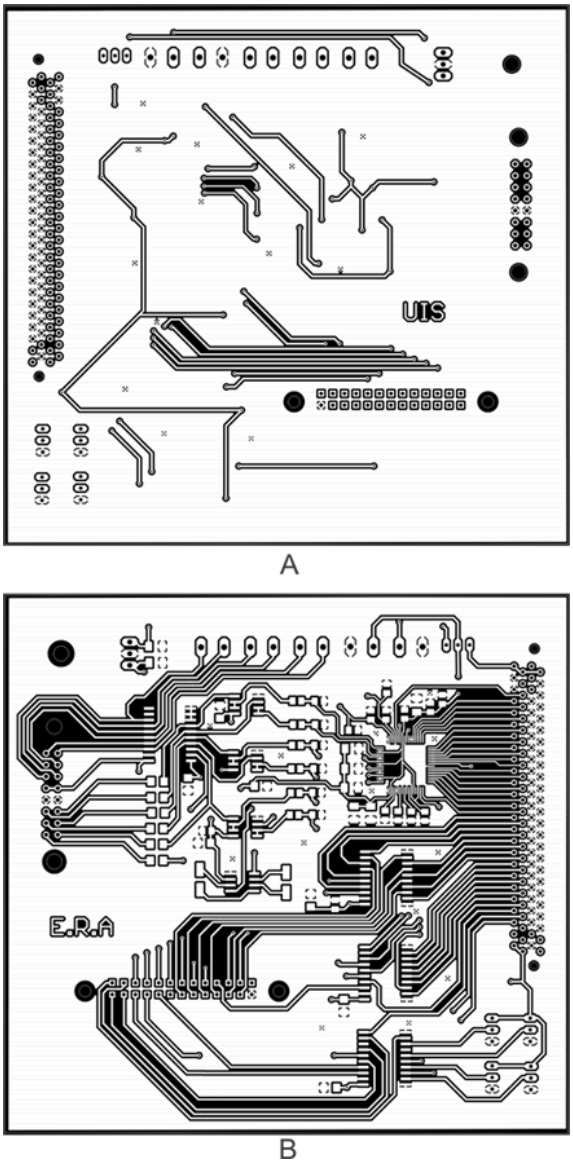

Figura D.1: Diseño PCB para la tarjeta Auxiliar. A) Plano Superior. B) Plano Inferior

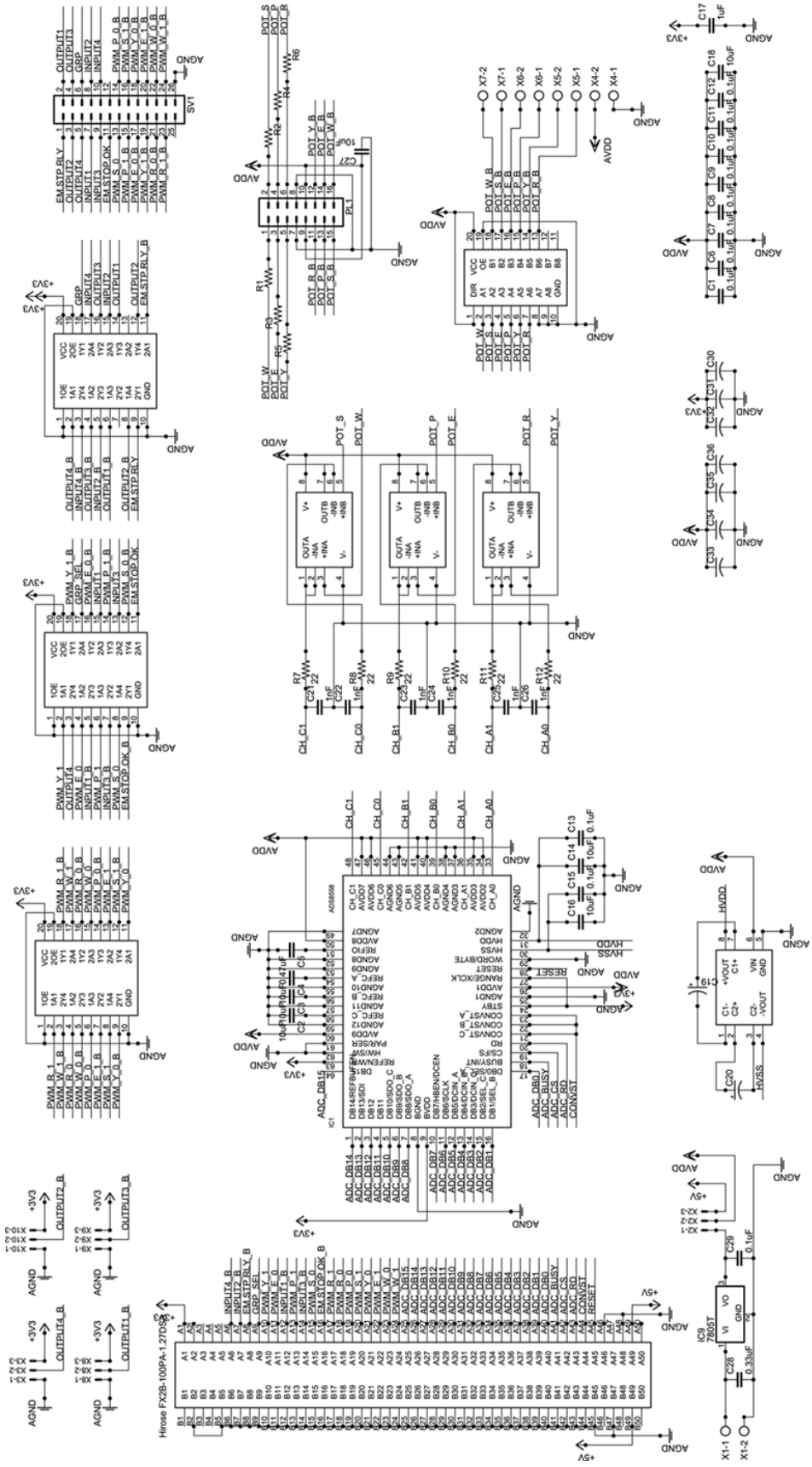

Figura D.2: Esquemático de la tarjeta auxiliar