UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño, implementación y verificación mediante modelos de cosimulación y hardware en el lazo de un control PI usando un FPGA

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

Sergio Mejía Serratos

DIRECTOR DE TESIS:
Ing. Alberto Ramiro Garibay Martínez

Ciudad Universitaria, Cd. Mx., 2018

JURADO:

PRESIDENTE: Dr. Paul Rolando Maya Ortíz

VOCAL: Ing. Alberto Ramiro Garibay Martínez

SECRETARIO: Dr. Hoover Mujica Ortega

1ER. SUPLENTE: M. en I. Mauro Gilberto López Rodríguez

2DO. SUPLENTE: M. en I. José Daniel Castro Díaz

ASESOR:

Ing. Alberto Ramiro Garibay Martínez

Agradecimientos

A mi esposa Claudia, por el infinito amor que me da, por ser la energía que me hace seguir adelante, por ser lo más importante para mi y la responsable de la culminación de este trabajo. Gracias por enseñarme lo que es determinación y voluntad.

A mi papá, que estuvo cerca de presenciar este gran momento de mi vida, que se perdió de la recompensa a todo su trabajo y empeño, por tantas cosas que hizo por mi. Por enseñarme a ser mejor cada día y a no tener miedo. Donde sea que estés gracias por todo.

A mi mamá, que estuvo al pendiente de mi educación, que siempre se ocupó en enseñarme lo más que podía, por tanto esmero y amor. Este trabajo es para ti y para mi papá.

A mi hermano Marco, que sin darse cuenta se convirtió en mi modelo a seguir y porque me dio uno de los mejores consejos de la vida.

A mi hermana Cecilia, que estuvo conmigo hasta el último día de la carrera, por cuidarme y darme tanto cariño, por enseñarme lo importante que es la dedicación.

A Gerardo, por su invaluable ayuda en todo momento, y por ser un gran amigo.

A Fernanda y Valeria, que con su llegada a la familia trajeron luz y felicidad.

A toda mi familia, que me apoyó y motivó para terminar este trabajo.

A mis amigos, que me soportan como soy y por brindarme su amistad incondicional.

A mis sinodales, por su eterna paciencia y por el apoyo brindado.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Actualmente la mayoría de los sistemas de control electrónico incorporan algún tipo de algoritmo de procesamiento, el cual permite una mejor ejecución de su tarea, una interacción más sencilla y/o eficiente con el usuario u otro tipo de sistemas. Mediante algoritmos de control embebidos estos sistemas proporcionan comunicaciones, transportación y automatización. La importancia de que estos algoritmos y sistemas electrónicos operen de forma satisfactoria se puede ver directamente relacionada con la seguridad de las personas. Por ejemplo, en los últimos años ha sido común que los fabricantes de automóviles realicen llamadas a revisión de sus vehículos debido a la detección de errores en los sistemas de control que podrían resultar en consecuencias fatales. Sin embargo, la detección de errores no siempre ocurre de forma oportuna, y es hasta que se tienen consecuencias que se encuentran dichas fallas. Una vez que un sistema de control se encuentra en funcionamiento en un producto final, los errores en el diseño podrían ser muy costos, ocasionando desde fuertes pérdidas económicas hasta pérdidas humanas. El desarrollar sistemas de control que incorporen capacidades complejas de procesamiento ha ido incrementando el uso del modelo de sistema durante el proceso de diseño, con el fin de entregar productos al mercado lo más rápido posible, pero que a su vez sean confiables. Actualmente existen compañías como National Instruments y The Mathworks Inc. que proveen herramientas de hardware y software permitiendo verificar el desempeño de un algoritmo durante la etapa de diseño y de esta forma corregir los errores que se pudieran presentar. El presente documento trata el proceso de diseño de un controlador PI utilizando la técnica de diseño basado en modelo y verificación mediante modelos de cosimulación y hardware en el lazo.

1.1. Diseño basado en modelo

El diseño basado en el modelo es un conjunto de técnicas orientadas al diseño de software embedido usado principalmente en aplicaciones de la industria aeroespacial y automotriz [2]. Este paradigma es diferente a la metodología tradicional. Mientras que en ésta se emplean estructuras complejas y código de software extenso, en el diseño basado en

el modelo se emplean los modelos matemáticos de la planta con el objetivo de obtener una representación de la dinámica del sistema que sea simulable. Esta nueva representación se traduce en un nuevo modelo representado mediante bloques o texto, el cual puede ser simulado, implementado y verificado. Las herramientas de verificación incluyen al propio simulador donde se está definiendo el modelo del sistema, herramientas de cosimulación y hardware en el lazo.
Algunas de las ventajas que este paradigma trae consigo son [3]:

- Se trabaja en un ambiente de diseño común, lo cual facilita la comunicación, análisis de datos y verificación del sistema entre grupos de desarrollo.
- Permite conocer errores en el diseño en etapas tempranas del proyecto, antes de que este se pruebe en la planta real.
- Permite el reuso del diseño, para actualizaciones o derivaciones de sistemas.
- Da la libertad de innovar experimentando con nuevas ideas.
- Con la integración de herramientas de software para modelado, simulación, generación de código y verificación se puede acelerar de forma considerable el desarrollo de sistemas embebidos.

1.1.1. Etapas del diseño basado en modelo

Las etapas para desarrollar un sistema embebido usando la metodología del diseño basado en el modelo es la siguiente [4]:

Definición del sistema

El primer paso es definir completamente el sistema. Si se está modelando un sistema grande que puede ser descompuesto en subsistemas se debe definir cada uno de ellos por separado tratándolo como una entidad independiente, para que al final se pueda realizar una integración y obtener finalmente la representación del sistema original. La definición de éste se da con la identificación de los siguientes componentes:

- Parámetros: Valores del sistema que permanecen constantes a menos que el diseñador los modifique.
- Estados: Variables en el sistema que cambian en el tiempo.
- Señales: Valores de entradas y salidas que cambian dinamicamente durante la simulación

Por cada subsistema identificado, el diseñador debe hacerse las siguientes preguntas:

- ¿Cuántas entradas tiene el subsistema?
- ¿Cuántas salidas tiene el subsistema?
- ¿Cuántos estados (variables) tiene el subsistema?
- ¿Cuáles son los parámetros (constantes) del subsistema?
- ¿Hay alguna señal intermedia (interna) en el subsistema?

Una vez que se han respondido estas preguntas, se tendrá una lista completa de los componentes del sistema por lo que se puede dar inicio a la etapa de modelado.

Modelado del sistema

El segundo paso es obtener el modelo de la planta. Una forma de modelar el sistema es mediante el uso del las ecuaciones matemáticas que describen su dinámica. Por cada subsistema se puede utilizar la lista de componentes identificados para describirlo en forma matemática. El modelo puede incluir:

- Ecuaciones algebraicas
- Ecuaciones lógicas
- Ecuaciones diferenciales para sistemas continuos
- Ecuaciones diferenciales para sistemas discretos

Es en este punto donde se obtiene la dinámica de la planta y donde se diseña el control, el cual debe llevar al sistema completo a cumplir con los requerimientos especificados.

Construcción del diagrama de bloques

Después de haber definido las ecuaciones matemáticas que describen cada subsistema, el siguiente paso es construir los diagramas de bloques de dichos modelos. Se deben construir los diagramas para cada subsistema y al final integrarlos en el modelo completo. Éste debe incluir cada componente que afecta al comportamiento del sistema, algoritmos, lógica de control, componentes físicos, núcleos de propiedad intelectual (IP cores) desarrollados en Matlab, C, HDL, herramientas especificas de modelado, etc.

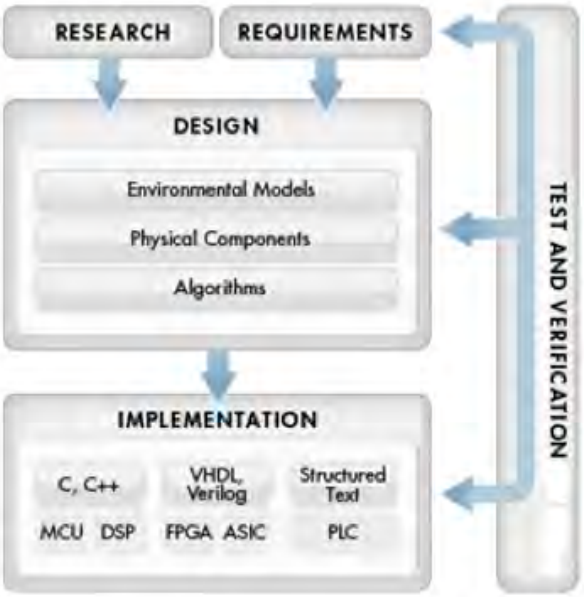

Figura 1.1: Diseño basado en modelo

Simulación

Una vez que se ha representado el modelo del sistema mediante diagramas de bloques y se ha diseñado el controlador, se puede simular el modelo y analizar los resultados. A través de las técnicas de simulación se pueden observar las ventajas y desventajas del diseño, siendo posible la corrección iterativa con el fin de refinar el diseño del modelo al nivel adecuado. La simulación nos permite analizar el rendimiento del sistema en condiciones que de otra forma consumirían mucho tiempo, serían riesgosas o caras.

Implementación

Cuando el diseño está listo para la implementación, se puede generar el código necesario automáticamente eliminando los errores de codificado a mano. Es posible la generación automática de código C, C++, HDL o texto estructurado desde el modelo para la realización de un prototipo rápido o para producción. El código generado puede ser optimizado y combinado con código escrito a mano.

Prueba y Verificación

Finalmente se debe verificar la implementación mediante las técnicas de cosimulación y hardware en el lazo es posible saber con anticipación si el controlador se comportará de forma adecuada cuando esté trabajando sobre el sistema real. La Figura 1.1 resume el flujo de trabajo utilizando el diseño basado en el modelo.

1.2. Objetivos

El objetivo de esta tesis es explorar algunas de las técnicas utilizadas para la generación de código automático y verificación de un algoritmo PI mediante modelos cosimulación y hardware en el lazo, las cuales suponen un ahorro de tiempo y esfuerzo durante la etapa inicial del desarrollo de un sistema de control, además de permitir la detección oportuna de errores en el diseño.
En particular, los objetivos del presente trabajo son los siguientes:

- A partir de un sistema modelado en Simulink, obtener un código VHDL de forma automática que permita describir a un FPGA.
- Mediante un modelo de cosimulación, verificar que el algoritmo PI cumple con los requerimientos del diseño.
- Mediante la técnica de hardware en el lazo, verificar el desempeño del algoritmo PI utilizando un FPGA en el lazo de control.

1.3. Organización de la tesis

La organización de los capítulos restantes de la tesis se detalla a continuación. El Capítulo 2 proporciona los conceptos básicos en la teoría de controles PID así como la descripción de las herramientas de software y hardware empleadas en este trabajo. En el Capítulo 3 se da una introducción a las herramientas de modelado, simulación y verificación así como al flujo de diseño característico. En el Capítulo 4 se aborda la integración de los conceptos revisados desde el Capítulo 2, mediante la implementación y verificación del algoritmo PI. Finalmente, en el Capiíulo 5 se encuentran las conclusiones del trabajo.

Capítulo 2

Preliminares

El propósito de un controlador es llevar a una planta a un estado deseado. Para eso se debe obtener la diferencia entre el estado actual de dicha planta y el estado al que se desea llevar, esta diferencia es conocida como el error del proceso [6]. Al conocer este error, el controlador aplicará diversas estrategias para llevar a la planta al estado deseado. Como ejemplo se puede suponer el caso donde el objetivo es llenar un vaso con agua como se muestra en la Figura 2.1 Su descripción, en lenguaje ordinario es muy simple: el usuario que llena el vaso con agua, mediante la observación del nivel alcanzado en el vaso, actúa sobre el grifo de modo de lo que va cerrando según se alcanza el nivel que estima oportuno. El proceso que tiene lugar lo describiríamos de la siguiente forma: el controlador (el que llena el vaso) compara el nivel alcanzado en el vaso con el nivel deseado, si existe discrepancia actúa sobre el grifo, con lo que se influye sobre el nivel alcanzado, que es de nuevo comparado (se trata de un proceso continuo) con el nivel deseado; según disminuya la discrepancia, se irá cerrando el grifo, hasta que al anularse esta, se cierra definitivamente.

Figura 2.1: Acción de control

2.1. Control PID

Las tres estrategias que aplica un control PID son las acciones proporcional, integral y derivativa, de tal forma que la salida del controlador es la acción combinada de estas tres acciones. Una forma de expresar la respuesta del control PID en el dominio del tiempo es mediante la siguiente ecuación, conocida como la forma paralela del controlador:

$$u(t) = K_p e(t) + K_i \int_0^\tau e(t)\ dt + K_d \frac{de(t)}{dt}$$

donde:

- $u(t)$ es la respuesta del controlador
- $e(t)$ es el error del proceso
- K_p es la ganancia proporcional
- K_i es la ganancia proporcional
- K_d es la ganancia proporcional

Cada uno de los términos de esta ecuación tiene un efecto específico sobre el estado de la planta [6]:

- Acción proporcional. La acción proporcional genera una salida que es proporcional al error. Sin embargo no asegura que el error en estado estacionario sea nulo.
- Acción Integral: La acción integral genera una salida que es proporcional al error acumulado. Esta señal permite eliminar el error en estado estacionario.
- Acción Derivativa: Genera una salida que es proporcional a la tasa variación del error. Esta acción sigue la variación de la variable controlada, de tal forma que cuando la variación de ésta es grande, el controlador actuará proporcionalmente al error generado por esta variación a fin de evitar que siga aumentando.

Al combinar estas tres acciones se obtiene la ley de control PID en forma paralela. Al tomar la transformada de Laplace de esta ecuación se llega a su representación en forma paralela en el domino de la frecuencia.

$$C(s) = K_p + \frac{K_i}{s} + K_d s.$$

La respuesta de un controlador PID continuo en el dominio del tiempo es expresada de la siguiente forma

$$U(s) = C(s)E(s),$$

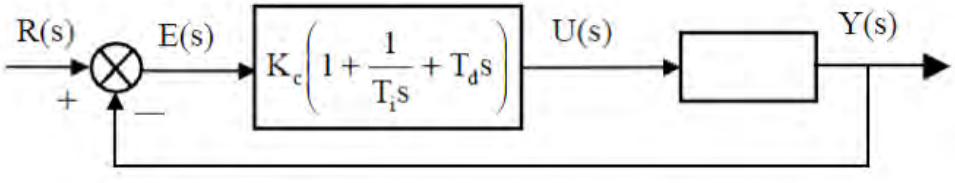

Figura 2.2: Diagrama de bloques del sistema de control con realimentación unitaria

donde $E(s)$ es el error del proceso y se define en el domino de la frecuencia como la diferencia entre el punto de ajuste R(s) y la salida de la planta o variable controlada Y(s)

$$E(s) = R(s) - Y(s).$$

Además, despejando $C(s)$ de la ecuación de respuesta del controlador, se obtiene la función de transferencia, que no es otra cosa mas que la relación entre la salida y la entrada

$$C(s) = \frac{U(s)}{E(s)}.$$

El diagrama de bloques del sistema de control con realimentación unitaria se muestra en la Figura 2.2.
La forma de representar a un controlador PID no es única, una forma alterna es mediante la llamada forma estándar o no interactiva del controlador. Dicha forma en el dominio del tiempo se expresa de la forma

$$u(t) = K_c\Big(e(t) + \frac{1}{T_i}\int_0^{\tau} e(t)\ dt + T_d\frac{de(t)}{dt}\Big) \tag{2.1}$$

donde:

- $u(t)$ es la respuesta del controlador
- $e(t)$ es el error del proceso
- K_c es la ganancia proporcional
- T_i es el tiempo integral
- T_d es el tiempo derivativo

Las formas paralela y estándar se relacionan mediante las siguientes ecuaciones

$$\begin{aligned} K_p &= K_c \\ K_i &= \frac{K_c}{T_i} \\ K_d &= K_c T_d \end{aligned}$$

Esta estructura del controlador también ha sido llamada como paralela no interactiva.

2.1.1. Métodos de sintonización

Una vez que se ha determinado el tipo de controlador que se va a implementar, se debe efectuar el ajuste de los parámetros (sintonización) para que la respuesta del sistema en lazo cerrado tenga unas características determinadas (criterio de sintonización). El ajuste de parámetros se convierte así en una tarea muy frecuente en plantas industriales, no solo en los trabajos de puesta en marcha, sino también cuando se detectan cambios sustanciales de comportamiento en el proceso controlado.
En las primeras aplicaciones de control PID, el ajuste se basaba únicamente en la propia experiencia del usuario o en métodos analíticos. En 1942, Ziegler y Nichols propusieron técnicas empíricas que tuvieron buena aceptación, y que han servido de base a métodos más recientes. Los métodos empíricos o experimentales de ajuste de parámetros están especialmente orientados al mundo industrial, donde existen grandes dificultades para obtener una descripción analítica de los procesos. Estos métodos constan fundamentalmente de dos pasos:

1. Estimación de ciertas características de la dinámica del proceso a controlar. Dicha estimación se puede efectuar en lazo abierto o en lazo cerrado.

2. Cálculo de los parámetros del controlador. Para ello se aplican las fórmulas de sintonización, que son relaciones empíricas entre los parámetros del controlador elegido y las características del proceso estimadas en el paso anterior.

El hecho de que estos métodos proporcionen sólo valores aproximados para los parámetros del controlador hace generalmente necesario un tercer paso (ajuste fino de los parámetros) mediante observación de la respuesta en lazo cerrado.Las diferencias entre los distintos métodos empíricos radican en la forma de combinar las técnicas de estimación y las fórmulas de sintonía.
La sintonía de controladores PID para procesos industriales está basada normalmente en especificaciones nominales sobre determinadas características de la respuesta del sistema en lazo cerrado a cambios bruscos en el punto de consigna o en la carga mientras que los valores de las ganancias proporcional, integral y derivativa determinan el correcto desempeño del controlador. Para cada sistema, existe un conjunto de estos parámetros que optimizan el funcionamiento del sistema, la selección no adecuada de parámetros puede producir un desempeño poco aceptable en el sistema y en casos extremos podría llevar el sistema a la inestabilidad. Los valores adecuados para estas constantes son aquellos en que el sistema exhiba un amortiguamiento crítico, encontrar analíticamente esos valores resulta difícil, pero existen dos métodos prácticos para encontrar una buena aproximación de estas constantes, los cuales se describen brevemente a continuación.

2.1.2. Método de la curva de reacción Ziegler-Nichols

Esta técnica se basa en el hecho de que la mayoría de los procesos industriales son estables en lazo abierto y que la respuesta del proceso a ciertas señales de entrada puede aportar en muchos casos información suficiente para poder diseñar un controlador satisfactorio. En particular, el método de Ziegler-Nichols en lazo abierto determina un ajuste de los parámetros del controlador en función de la respuesta del sistema a un escalón en la entrada del mismo. En la Figura 2.3 se observa la salida de un sistema dinámico frente a un incremento en la entrada del mismo aplicado en el instante $t = 0$.

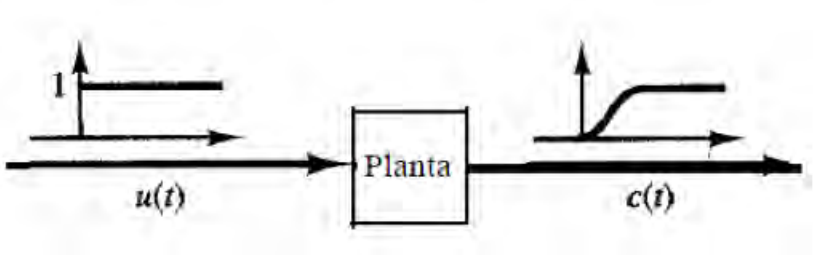

Figura 2.3: Sistema sometido a una entrada escalón unitario

La curva con forma de S se caracteriza por dos parámetros: el tiempo de retardo L y la constante de tiempo T. El tiempo de retardo y la constante de tiempo se determinan dibujando una recta tangente al punto de inflexión de la curva con forma de S y determinando las intersecciones de esta recta tangente con el eje del tiempo y con la línea $c(t) = k$, como se muestra en la Figura 2.4.

Ziegler-Nichols sugirieron establecer los valores de K_p, T_i, T_d de acuerdo con la fórmula que se muestra en la Tabla 2.1.

Controlador	K_p	K_i	K_d
P	T/L	-	-
PI	$0.9T/L$	$L/0.3$	-
PID	$1.2T/L$	$2L$	$0.5L$

Tabla 2.1: Ganancias respecto a los parámetros T y L

De esta forma, las ganancias de cada acción de control pueden ser obtenidas conociendo los parámetros de tiempo de retardo L y de constante de tiempo T.

2.1.3. Método oscilaciones sostenidas

Ciertas características dinámicas de los procesos también se pueden determinar a partir de su respuesta en frecuencia. Existen varios métodos experimentales para la determinación indirecta de un punto de la respuesta en frecuencia, concretamente para determinar

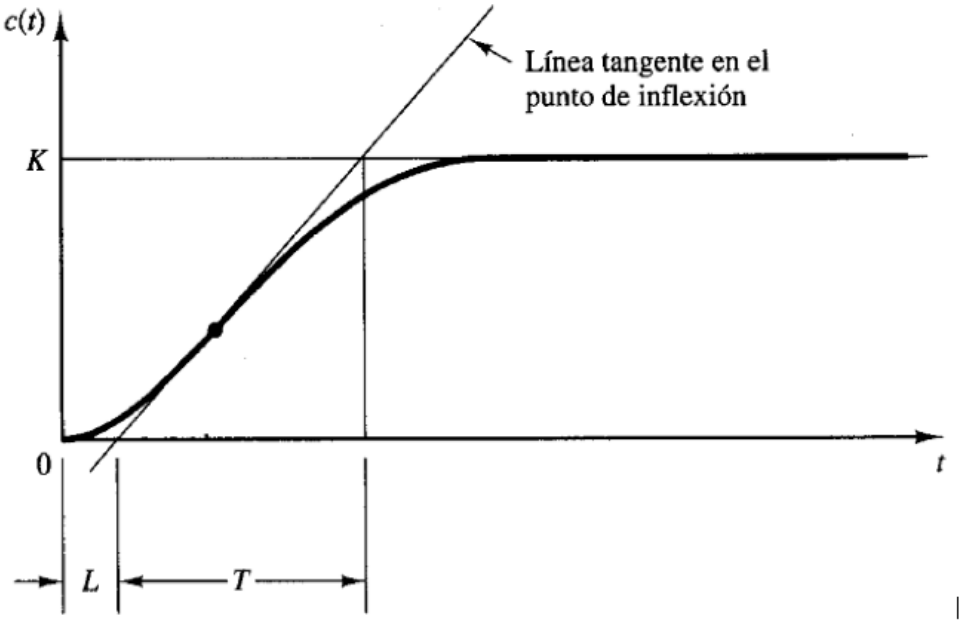

Figura 2.4: Respuesta del sistema a una entrada escalón unitario

la ganancia última K_u y el periodo de oscilación mantenida P_{rc}, definidos respectivamente como: la ganancia de un controlador proporcional a partir de la cual el sistema en lazo cerrado deja de ser estable, y el periodo de la oscilación que se consigue con ese valor de ganancia. Ziegler y Nichols, en función del valor de dicho punto de frecuencia, proporcionan el valor de los parámetros del controlador PID. Uno del los métodos más conocidos para la obtención de dichas características de frecuencia es el método de las oscilaciones sostenidas, el cual se realiza de la siguiente forma:

1. Se cierra el lazo de control con el controlador en modo proporcional únicamente.

2. Con la ganancia proporcional K_p a un valor arbitrario, se provocan pequeños cambios bruscos en el punto de consigna y observar la respuesta del sistema.

3. Se aumenta o disminuye K_p hasta conseguir en el paso anterior que el sistema oscile con una amplitud constante como se muestra en la Figura 2.5. Anotar el valor de la ganancia proporcional en ese instante como K_{cr} y medir el periodo de la oscilación mantenida P_{cr}.

Ziegler y Nichols sugirieron que se establecieran los valores de los parámetros K_p, T_i y T_d de acuerdo a la Tabla 2.2.

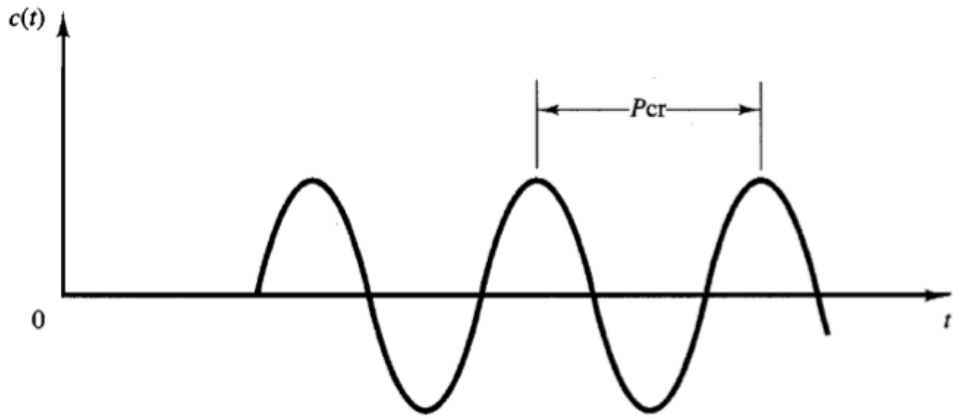

Figura 2.5: Respuesta oscilatoria de un sistema de control

Controlador	K_p	K_i	K_d
P	$0.5K_{cr}$	-	-
PI	$0.45K_{cr}$	$P_{cr}/1.2$	-
PID	$0.6K_{cr}$	$0.5P_{cr}$	$0.125P_{cr}$

Tabla 2.2: Ganancias respecto a los parámetros K_{cr} y P_{cr}

2.2. VHDL y FPGA

Retomando el sistema de control mostrado anteriormente en el que se describe el proceso de llenado con agua de un vaso, el objetivo ahora es encontrar un elemento que permita implementar el algoritmo de control de forma automática. En este punto se busca un dispositivo que reemplace a la persona que realiza la tarea de control. Actualmente existen diferentes formas y tecnologías para realizarlo, sin embargo en el caso de este trabajo el enfoque se centra en los controladores electrónicos digitales. Esta clasificación incluye varios dispositivos que pueden realizar la tarea de control, tales como computadores personales, controladores lógicos programables (PLC), procesadores digitales de señales (DSP), microcontroladores y dispositivos lógicos programables (PLD). Dentro de la categoría de los PLD hay varias opciones tales como los dispositivos de lógica compleja programables (CPLD), arreglos de compuertas programables en campo (FPGA) y los circuitos integrados de aplicación específica (ASIC). Cada uno de estos dispositivos cuenta con características únicas que los hacen ideales para aplicaciones específicas.
Los FPGA han experimentado un crecimiento en popularidad en los últimos años debido a los bajos costos de desarrollo de productos, la flexibilidad de estos dispositivos y las nuevas herramientas de software que permiten reducir el tiempo de diseño de circuitos digitales.

Los FPGA son enormes campos de compuertas programables en donde las señales eléctricas pueden viajar a través de diferentes rutas del circuito pudiendo ser procesadas por diferentes secciones de hardware de forma paralela [4]. Son inherentemente paralelos, por lo que las diferentes secciones de procesamiento no tienen que competir por los mismos recursos. Los diseñadores pueden mapear sus diseños automáticamente al FPGA, esto permite al usuario crear cualquier número de circuitos los cuales procesarán las señales eléctricas de forma paralela.

En este trabajo de tesis se emplea un FPGA como dispositivo contenedor del algoritmo de control. A continuación se describe brevemente los elementos necesarios para realizar un diseño basado en FPGA así como el proceso basado en la descripción del hardware.

2.2.1. Lenguaje de descripción de hardware (HDL)

El primer elemento para diseñar circuitos digitales basados en FPGA puede ser un lenguaje de descripción de hardware (HDL) el cual es usado para describir circuitos digitales. Los lenguajes HDL también pueden ser usados para describir algoritmos no sintetizables tales como rutinas de análisis de archivos ya que un lenguaje HDL no es un lenguaje de programación de software que se use para generar programas que son traducidos a instrucciones de máquina que luego son ejecutados por una computadora [1]. Un lenguaje HDL tiene un soporte sintáctico y semántico diseñado para modelar el comportamiento temporal y la estructura espacial del hardware. Dicho de otra manera, un lenguaje de programación de software crea un conjunto de instrucciones que deben ser ejecutadas en una computadora, mientras que el HDL describe a la computadora misma. VHDL es uno de los lenguajes de descripción de hardware más usados, es el acrónimo de *VHSIC Hardware Language Description*, donde, VHSIC es el acrónimo de *Very High-Speed Integrated Circuit*. Fue originalmente desarrollado por el departamento de defensa de los Estados Unidos de América a inicios de la década de los 80 y después fue transferido al IEEE (Instituto de Ingenieros Eléctricos y Electrónicos). El lenguaje es formalmente definido por el estándar IEEE 1076. Este estándar fue ratificado en 1987, referido como VHDL 87 y ha sido revisado en varias ocasiones siendo las más empleadas las revisiones VHDL93 y VHDL200X.
VHDL fue desarrollado para describir y modelar sistemas digitales a diferentes niveles de abstracción, derivando una elevada complejidad del lenguaje. De todo el conjunto de instrucciones que conforman el VHDL, solo un subconjunto de este es sintetizable, es decir, solo algunas instrucciones se pueden representar mediante hardware.

2.2.2. FPGA

Un FPGA (arreglo de compuertas programables en campo) se puede definir como un dispositivo electrónico que puede ser reprogramado para implementar cualquier circuito de lógica combinacional y/o secuencial. Este dispositivo presenta una estructura regular

de celdas lógicas e interconexiones que están bajo completo control del diseñador. Los fabricantes más conocidos de estos dispositivos son las campañas Xilinx Inc. y Altera Corp. En la Figura 2.6 se muestran los empaquetados de algunos FPGAs.

Figura 2.6: FPGA manufacturados por Xilinx y Altera

Arquitectura de un FPGA

La arquitectura de un FPGA no es única, cada fabricante de FPGAs tiene su propia arquitectura, incluso las diferencias entre las familias de FPGA de un mismo fabricante son notables. Sin embargo, en términos generales, los FPGA presentan una variación de la estructura mostrada en la Figura 2.7. Esta arquitectura consiste de tres elementos básicos:

- Bloques de lógica configurable.
- Bloques de E/S configurables.
- Interconexiones configurables.

Los bloques de lógica configurable mostrados en la Figura 2.8, conocidos como CLB (Configurable Logic Block) por Xilinx o LAB (Logic Array Block) por Altera, son los bloques que contienen la lógica del FPGA. Los bloques configurables están organizados en una red dentro del FPGA y están conectados a los recursos de interconexión. Los CLB consisten de un cierto número de células lógicas llamadas *slices* por Xilinx, LE (Logic Element) para la familia Cyclone III o ALM (Adaptive Logic Modules) para la familia Stratix IV por Altera. El número y funcionalidad de estas células lógicas que contiene cada FPGA varía según su modelo y marca. Específicamente hablando de los FPGA de la familia Spartan 6 de Xilinx, un CLB contiene 2 Slices, cada Slice contiene 4

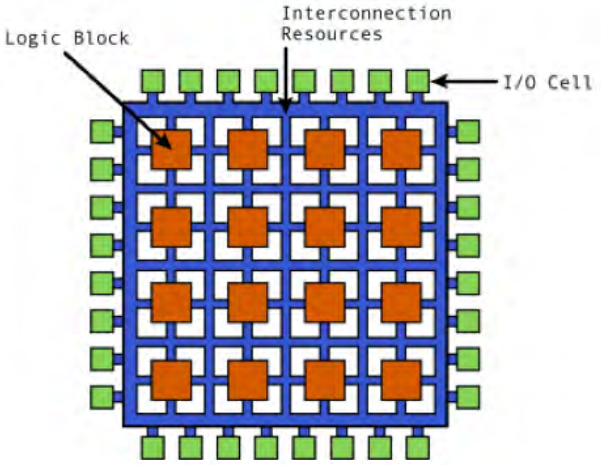

Figura 2.7: Arquitectura básica de un FPGA

LUTS, 8 Flip-Flops y multiplexores, los cuales entre otras cosas, permiten generar salidas síncronas o asíncronas. Algunos Slices contienen lógica adicional la cual permite a las LUT comportarse como memoria RAM distribuida o como registros de desplazamiento. Adicionalmente, pueden contar con un bloque de lógica de acarreo, la cual permite realizar adiciones o substracciones de manera eficiente. La familia Spartan 6 cuenta con tres tipos diferentes de Slices, los cuales son:

1. SliceM (25 %): Ofrece lógica de acarreo, multiplexores de funciones ampliadas y comportamiento de RAM distribuida o registro de desplazamiento (SRL).

2. SliceL (25 %): Ofrece lógica de acarreo, multiplexores de funciones ampliadas

3. SliceX (50 %): Ofrece optimización para funciones lógicas.

Los bloques configurables de E/S cuyo esquemático se muestra en la Figura 2.9, son usados como una interfaz que permite el intercambio de señales entre dispositivos externos y los bloques de lógica interna. Estos bloques están conformados por buffers de entrada y buffers de salida con control tri-estado y colector abierto. Típicamente tienen un resistor de pull-up en la salida y en algunas ocasiones, resistores de pull-down los cuales pueden ser utilizados para finalizar señales y buses sin que sea necesario agregar resistores externos al chip. La polaridad de una salida usualmente puede ser programada como activa alta o activa baja, así mismo, la velocidad de respuesta de la salida puede se programada para tiempos de levantamiento y caída rápidos o lentos.

Las interconexiones configurables como la que se muestra en la Figura 2.10 son líneas largas que pueden ser usadas para interconectar CLBS que están físicamente lejos uno del otro sin introducir un elevado tiempo de retardo. Estas líneas largas pueden ser usadas como buses de datos dentro del chip. También hay líneas cortas que son usadas para

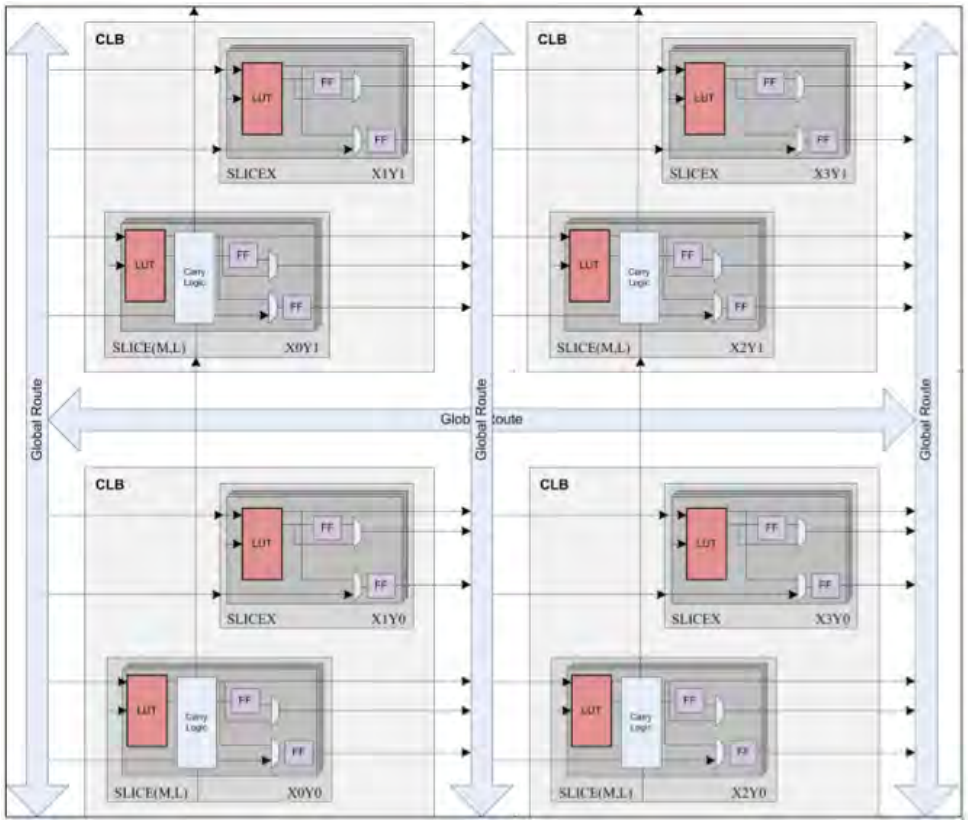

Figura 2.8: Bloques de lógica configurable

interconectar CLB que están físicamente cerca uno del otro. Los transistores se utilizan para activar o desactivar las conexiones entre las diferentes líneas, además, se cuenta con varias matrices de interruptores programables que permiten conectar las líneas largas con las líneas cortas permitiendo formar estructuras muy flexibles de comunicación. Se usan también Buffers Tri-Estado para conectar varios CLBS a una línea larga y crear un bus. Además existe un conjunto de líneas largas especiales llamado globales de reloj, las cuales son especialmente diseñadas para tener una baja impedancia y de esta forma proporcionar tiempos de propagación rápidos. Estas están conectadas a buffers de reloj y a cada elemento sincronizado en cada CLB. De esta forma la señal de reloj es distribuida a través de FPGA, asegurando un sesgo mínimo entre las señales de reloj que llegan a los diferentes Flip-Flops dentro del chip. En los FPGA la mayor parte del retraso de propagación de las señales dentro del chip ocurre en las interconexiones, debido a que estas se encuentran, así como los CLBS, fijas al chip. Con el fin de conectar un CLB a otro en una parte diferente del chip, es frecuente requerir una conexión con muchos transistores y matrices de interruptores, cada uno introduciendo un retraso adicional. Dependiendo del FPGA, este puede tener recursos adicionales, tales como bloques de memoria, bloques

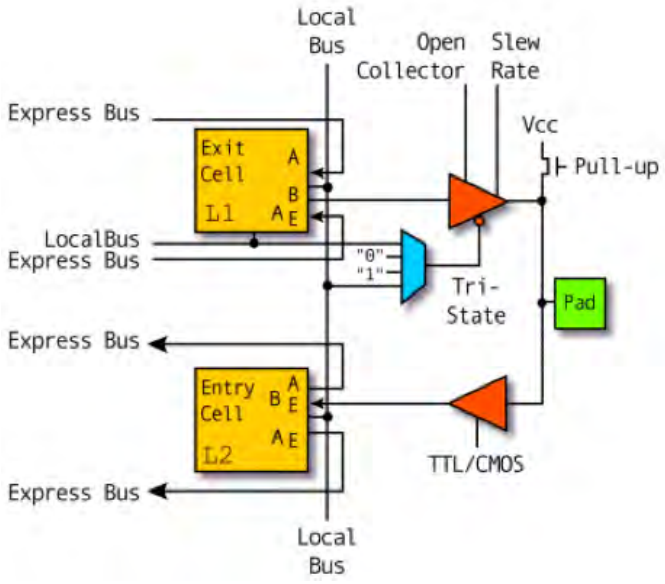

Figura 2.9: Bloque configurable de E/S

DSP o unidades de aritmética y lógica (ALU).

2.3. Diseño con VHDL y FPGA

El flujo de diseño básico comienza con la descripción del circuito, usualmente mediante un lenguaje de descripción de hardware tal como VHDL. Seguida por un proceso de síntesis, en donde la descripción es transformada en una especificación a nivel de compuertas primitivas. Una vez que la descripción ha sido verificada a través de la simulación, la tecnología de mapeado traslada las compuertas primitivas a una red de bloques de lógica programable. En seguida el proceso de colocación y enrutado se encarga de colocar y conectar cada uno de los CLBs dentro del FPGA. Los diversos fabricantes proporcionan herramientas de software que facilitan el flujo de diseño usando lenguajes de descripción de hardware o interfaces para la captura de esquemáticos. Mediante la captura del esquemático, el diseñador emplea herramientas de software con interfaces gráficas que le permiten especificar la posición, tipo y cantidad exacta de compuertas requeridas y como están interconectadas entre sí.
Existen 4 pasos básicos durante la captura del esquemático:

1. Construir el circuito usando las librerías de componentes específicos para el dispositivo a emplear.

2. Conectar las compuertas.

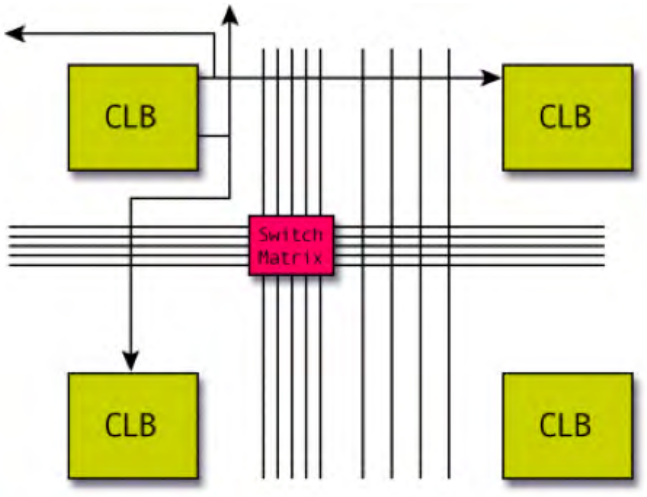

Figura 2.10: Interconexión configurable

3. Agregar y etiquetar las entradas y salidas del circuito.

4. Generar un *netlist*. Un *netlist* es un archivo que contiene una descripción textual equivalente del circuito, la cual es generada por herramientas de software de diseño llamadas sintetizadorores. El *netlist* es una forma compacta de describir el arreglo de compuertas y las conexiones entre ellas, la cual permite que cualquier otra herramienta de software que pueda leer este archivo sea capaz de determinar la función que realiza el circuito.

Sin embargo esta forma de describir circuitos digitales tiene algunos inconvenientes, cuando el circuito es demasiado grande, la captura del esquemático se vuelve compleja y tardada, además, si el dispositivo a emplear se cambia, esto implicará que el diseño deberá ser capturado nuevamente empleando los recursos del nuevo dispositivo. Otra forma de describir circuitos digitales es mediante los lenguajes de descripción de hardware tales como ABEL, AHDL, Verilog, VHDL, entre otros siendo estos últimos dos los más populares. La idea básica de usar un lenguaje de alto nivel es describir el circuito en forma textual en vez de una descripción gráfica de compuertas a bajo nivel. En estos lenguajes el término *B*ehavioural es usado por los diseñadores debido a que la forma de describir un circuito es realizada mediante la descripción de la función o comportamiento del sistema en lugar de la descripción de las compuertas empleadas y sus interconexiones. De esta forma la descripción de sistemas grandes puede realizarse a través de unas pocas líneas de código. El archivo donde reside la descripción textual contiene toda la información necesaria para definir el circuito. Otra ventaja es que esta descripción es totalmente independiente del dispositivo a emplear, lo cual quiere decir, que el mismo código puede utilizarse para describir diferentes dispositivos.

Una vez descrito el funcionamiento o comportamiento del circuito, una herramienta de software llamada herramienta de síntesis realiza una tarea intensiva para generar todas las compuertas necesarias y las interconexiones entre ellas detalladas en la descripción de alto nivel, este proceso es dependiente del dispositivo a emplear, por lo que esta herramienta usa los recursos propios de cada dispositivo a fin de obtener una descripción adecuada. Aunado a esto, el diseñador puede especificar algunos criterios de optimización, los cuales la herramienta de síntesis tomará en cuenta durante la selección o mapeo de compuertas. Dentro de estos criterios se encuentran la optimización del diseño completo para utilizar el menor número de compuertas (optimización de área), optimizar cierta sección del diseño para una velocidad de operación rápida (optimización de velocidad), seleccionar la mejor configuración de compuertas para minimizar la potencia consumida (optimización de potencia). El proceso de síntesis concluye con la generación del archivo *n*etlist, que se empleará en posteriores etapas para su implementación en el dispositivo.

Después de completado el diseño del circuito, es necesario saber si este tiene el funcionamiento esperado. Este proceso se denomina verificación del diseño, en el cual se usa una herramienta de software conocida como simulador HDL. Algunos simuladores HDL pueden trabajar con la descripción realizada en algún lenguaje HDL evitando el proceso de síntesis o bien, pueden trabajar con el archivo *n*etlist generado por alguna herramienta de síntesis. Adicionalmente, se requiere un archivo que contenga un patrón de entradas o señales que servirán para probar el funcionamiento del circuito. Este patrón de entradas es conocido como *t*estbench, el cual es escrito también en un lenguaje de descripción de hardware, el código contenido en él no es sintetizable, es decir, no puede ser traducido a hardware.. A partir de esta información, el simulador HDL podrá determinar las salidas que proporciona el circuito a las entradas especificadas y el diseñador decidirá si estas cumplen con el funcionamiento especificado para el diseño. Esta simulación es conocida como simulación funcional, debido a que únicamente se revisa si el circuito genera las combinaciones correctas de unos y ceros, este proceso consume la mayor cantidad de tiempo durante el diseño de un sistema digital. En este punto se cuenta con el archivo *n*etlist que describe lo describe usando compuertas específicas para un dispositivo y que se ha probado que genera las combinaciones de unos y ceros adecuadas. Es momento de introducir el diseõ contenido en el archivo en el FPGA, este proceso es conocido como implementación en el dispositivo. Este proceso está constituido por varias etapas, los cuales se describen brevemente a continuación:

- La etapa de traducción es realizada por varios programas para importar el diseño especificado en el archivo *n*etlist y prepararlo para su acomodo en el dispositivo, algunas del las tareas que se realizan son las de optimización, traducción a los elementos físicos del dispositivo, revisión de reglas de diseño específicas del dispositivo (por ejemplo, que el diseño no se exceda el número de buffers de reloj disponibles en el dispositivo). Es en esta etapa donde la información tal como el modelo del dispositivo, tipo de encapsulado, grado de velocidad y otras características específicas del dispositivo son requeridas. Usualmente se termina con un reporte de resultados

generados por estas herramientas, donde advertencias, errores, utilización de recursos de E/S son proporcionados al diseñador. De esta forma es posible saber si se ha seleccionado el dispositivo adecuado o si es necesario un dispositivo de menor o mayor capacidad.

- La siguiente etapa dentro del proceso de implementación es la colocación y enrutado. La tarea de colocación es el proceso de selección de módulos específicos o bloques de lógica en el FPGA donde las compuertas residirán. El enrutado, es el proceso de realizar la interconexión física entre dichos bloques. Este proceso es realizado automáticamente, pero algunos proveedores proporcionan herramientas de software que permiten realizar el proceso de colocación y enrutamiento de las partes más críticas de forma manual y obtener un mejor rendimiento que con herramientas automáticas. Estas dos herramientas de software requieren la mayor cantidad de tiempo para completar su trabajo, debido a que es muy complejo determinar la ubicación de cada componente en diseños grandes, asegurarse de que todos los elementos queden bien conectados y que cumplan el rendimiento especificado. Una herramienta de software relacionada en este proceso, es la llamada herramienta de colocación y enrutado controlada por tiempo, la cual permite especificar criterios de tiempo que serán usados durante la disposición del dispositivo.

- La etapa siguiente en el análisis estático de tiempo, ésta provee información de tiempo acerca de las rutas del diseño la cual es muy exacta, puede ser vista de diferentes maneras, clasificándolas desde el retraso más largo al más corto. En este punto se puede usar la información detallada generada en este proceso durante la disposición de las compuertas en el FPGA y simular nuevamente el circuito. En esta simulación, el diseñador se enfoca en observar que la salida de unos y ceros no solamente sea la correcta, sino que ocurra en el tiempo adecuado, esta simulación se conoce como simulación de tiempo. El proceso de obtener la información de tiempo del sistema y utilizarlo en el simulador HDL se conoce como anotación de regreso. En este caso, la información de tiempo refleja los retrasos en los bloques de lógica así como en las interconexiones.

- La etapa final es la descarga o programación del FPGA. Como su nombre lo indica, este proceso descarga la información de la configuración del dispositivo a un dispositivo de memoria. Esta información llamada *b*itstream, contiene toda la información que define la lógica y las interconexiones del diseño. Debido a que los dispositivos FPGA pierden su configuración cuando se suspende el suministro de energía eléctrica, el *b*itstream debe de ser almacenado en un lugar que pueda retener dicha información aún en la ausencia de energía eléctrica. Un lugar común para almacenar la información es una memoria PROM.

El término programación es usado para programar todos los dispositivos no volátiles incluyendo las memorias PROM. El proceso de programación realiza la misma función

que el proceso de descarga, con la excepción de que la información de configuración es retenida aún después de cortar el suministro de energía. La forma más común de descargar o programar estos dispositivos es a través del protocolo JTAG (Joint Test Advisory Group) conocido también como IEEE/ANSI estándar 1149.1-1990. El cual define una serie de reglas que facilitan la prueba, programación y depuración de algunos dispositivos electrónicos. La etapa final es la depuración del sistema, en este punto el dispositivo ya está trabajando, pero se tiene que revisar que el dispositivo trabaje adecuadamente el la tarjeta donde está instalado. Los errores encontrados aquí implican que el diseñador realizó una suposición en las especificaciones del dispositivo que no es correcta o que no consideró algunos aspectos de las señales requeridas de o hacia el dispositivo lógico

2.4. Kit de evaluación de Spartan SP601

El kit de evaluación Spartan SP601mostrado en la Figura 2.11, es un sistema electrónico desarrollado por Xilinx Inc. Constituido principalmente por la tarjeta de desarrollo SP601, este diseño proporciona todos los elementos necesarios para que los desarrolladores de hardware y software puedan crear y evaluar programas mediante un FPGA. Estos elementos, tales como el sistema de suministro eléctrico, reloj, sistemas de comunicación USB y Ethernet entre otros, permiten que la prueba de diseños digitales se realice de forma rápida y sencilla.

Figura 2.11: Kit de evaluación Spartan

La tarjeta de desarrollo Spartan SP601 es parte de este kit y se muestra en la Figura 2.12. Está basada en un FPGA de la familia SPARTAN 6, la cual se caracteriza por ofrecer un balance óptimo entre costo, potencia, rendimiento y soporte de herramientas de desarrollo para crear productos orientados al área de consumo, automotriz, vigilancia,

comunicación inalámbrica y otros mercados donde el costo del producto sea un factor importante.

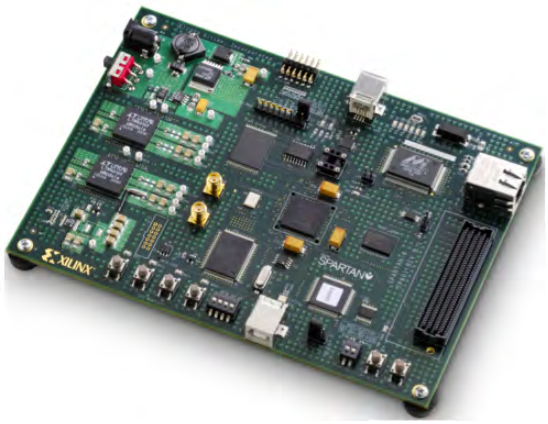

Figura 2.12: Tarjeta de desarrollo Spartan

El FPGA de esta tarjeta es un Spartan 6 XC6SLX16-CS324 y en la tabla 4.1 se muestran algunas de sus características más importantes.

FPGA	XC6SLX16
Slices	2,278
CLB	18,224
RAM máxima distribuida (Kb)	136
Bloques de RAM (18Kb)	32
DSP Slices	32
E/S	232
Kit de desarrollo	Spartan 6 SP601

Tabla 2.3: Características de la tarjeta

La tarjeta de desarrollo SP601 incluye memorias DDR2 y FLASH, conexión Ethernet de 1Gbps, UART y puertos de E/S de propósito general. Además, incluye un conector llamado FMC (FPGA Mezzanine Card), el cual permite añadir funcionalidad a la tarjeta mediante la interconexión de tarjetas de expansión especialmente diseñadas con este conector. Entre las tarjetas más comunes se encuentran las de conversión AD/DA, conectividad serie, procesamiento de imágenes e interfaces para motores. El kit incluye una fuente de 5 V, dos cables USB, un cable para conexión Ethernet y el software necesario para capturar diseños e implementarlos en el FPGA.

Capítulo 3

Herramientas para el modelado, simulación y verificación

Diseñar un algoritmo para el procesamiento de señales usando HDL consume mucho tiempo y es propenso a errores. Determinar el algoritmo correcto para el procesamiento de señales antes de capturar el código HDL toma mucho tiempo, pero en general, es más rápido encontrar errores, lo cual es siempre deseable en el ciclo de diseño.
Para modelar sistemas basados en FPGA se necesitan herramientas que puedan producir modelos basados en aritmética de punto fijo con bits verdaderos y ciclos exactos. Existen muchas herramientas tales como C/C++, LabVIEW, Matlab, Simulink que ofrecen esta caracterÍstica. En este proyecto, se usan principalmente las herramientas de software desarrollados por The Mathworks Inc., Matlab y Simulink. La tendencia en la industria es la de usar herramientas de generación automática de código HDL. Hay una gran cantidad de este tipo de herramientas disponibles, las cuales son capaces de generar código VHDL o *v*erilog sintetizable e independiente del dispositivo a partir de modelos en Simulink, por ejemplo, HDL Coder de The Mathworks Inc. o Synplify DSP de Synplicity Inc. Además hay herramientas que producen código dependiente del dispositivo desde un modelo de Simulink, entre estas herramientas se encuentran el Altera DSP Builder y el Xilinx System Generator.
A continuación se da una breve descripción de las herramientas empleadas en este trabajo de tesis.

3.1. Herramientas de software

3.1.1. MatLab

Matlab es en lenguaje de alto nivel y un entorno interactivo para computación numérica, visualización y programación desarrollado por The Mathworks Inc. Con Matlab se puede analizar datos, desarrollar algoritmos, crear modelos y aplicaciones. Incluye un gran

número de funciones predefinidas que permiten explorar diferentes enfoques y alcanzar una solución más rápido que con los lenguajes de programación tradicionales tales como C/C++ o Java.

3.1.2. Simulink

Simulink es un entorno de programación mediante diagramas de bloques, el cual permite la simulación multidominio y diseño basado en modelos. Soporta el diseño de sistemas, simulación, generación automática de código y la continua prueba y verificación de sistemas embebidos. Simulink provee un editor gráfico, librerías de bloques personalizables, solucionadores de ecuaciones los cuales permiten el modelado y simulación de sistemas dinámicos. Está integrado con Matlab, permitiendo la incorporación de algoritmos escritos en Matlab en el modelo. Además permite la exportación de los resultados de la simulación a Matlab para un análisis adicional.

3.1.3. ModelSim

ModelSim es una herramienta de software desarrollada por Mentor Graphics, la cual administra un entorno que permite editar, compilar, simular y depurar diseños de sistemas digitales descritos en VHDL, Verilog y SystemC.

3.1.4. HDL Coder

EL HDL Coder es una herramienta de software desarrollada por The Mathworks Inc. que permite generar código Verilog o VHDL portable y sintetizable a partir de un modelo en Simulink, una función en Matlab o un diagrama de estados en Stateflow. El código HDL generado puede ser optimizado para cumplir con requerimientos de área, potencia o velocidad, además, puede ser simulado usando herramientas de software de terceros, tales como Mentor Graphics ModelSim y Xilinx Isim. El HDL Coder cuenta con un asistente que guía al usuario durante la generación de código y programación del FPGA, conocido como HDL Workflow Advisor. A través de la integración de herramientas de software de terceros tales como ISE Web Pack de Xilinx y ModelSim de Mentor Graphics, El HDL Workflow Advisor proporciona un flujo de trabajo particular, dependiendo de la tecnología seleccionada. Dentro de los flujos de trabajo soportados, se tienen:

- Generic ASIC/FPGA.
- FPGA-in-the-Loop.
- FPGA Turnkey.
- Customization for the USRP Device.

Cada flujo de trabajo está formado por una serie de tareas específicas, las cuales están agrupadas en categorías, esto permite seguir una secuencia sobre el flujo de diseño. Cada tarea realiza un paso dentro del flujo de trabajo. Al finalizar la ejecución, se genera un reporte del proceso. Si la tarea falló, se proporcionará la información necesaria para modificar el modelo y completarla. Una vez finalizadas todas las tareas se obtendrá un reporte generado por la herramienta de síntesis empleada. Si los resultados obtenidos no cumplen con las especificaciones requeridas, se puede modificar el modelo original, cambiar la implementación o usar una opción diferente de generación de código, de esta forma se puede ir depurando el diseño hasta que los requerimientos sean alcanzados.
El HDL Workflow Advisor para FPGA-in-the-Loop proporciona las siguientes tareas:

- **Set Target**: Las tareas de esta categoría permiten seleccionar el dispositivo que se programará, en este caso una Tarjeta Xilinx Spartan 601, así como la herramienta de síntesis a utilizar.
- **Prepare Model For HDL Code Generation**: Las tareas de esta categoría permiten determinar si es posible obtener una descripción HDL a partir del modelo. En caso de que el modelo no pueda ser sintetizado, esta herramienta brindará un informe de los elementos no sintetizables así como una ayuda para resolver los conflictos de compatibilidad. Las revisiones que se hacen son la revisión de parámetros globales, comprobación de que no existan lazos algebraicos y comprobación de compatibilidad de los bloques usados en el modelo.
- **HDL Code Generation**: En esta sección, se encuentran las tareas que nos permiten realizar la configuración de parámetros de generación de *t*estbench, generación de código HDL, reportes de trazabilidad y recursos consumidos así como, la configuración del modelo de cosimulación.
- **FPGA-in-the-Loop Implementation**: En esta categoría se implementan todas las fases de configuración del FPGA-in-the-Loop, tales como:
  - Generación del bloque FIL
  - Síntesis
  - Mapeo de lógica
  - Ubicación y enrutamiento (Place-and-route)
  - Generación del archivo de programación
  - Configuración del canal de comunicación

3.2. Herramientas de verificación

Para una verificación rápida del diseño, el HDL Coder es capaz de generar *t*estbenches y modelos de cosimulación. Éstoso, junto con el código VHDL generado, permiten la

verificación del diseño en cualquier simulador HDL, tal como el Xilinx Isim. Por otra parte, con la incorporación del HDL Verifier la cual es una herramienta de software desarrollada por The Mathworks Inc. Es posible establecer una interface entre Matlab y Simulink con simuladores HDL y tarjetas compatibles con la verificación de FPGA en el lazo. La estimulación de los modelos de cosimulación se realiza mediante las mismas entradas de prueba que se usan en Matlab o Simulink, eliminando de esta forma la necesidad de crear los *t*estbenches.

3.2.1. Verificación del modelo de cosimulación mediante un simulador HDL

La cosimulación mediante el uso de un simulador HDL permite verificar que el código HDL generado coincida con el algoritmo en Matlab o el modelo el Simulink proporcionado visibilidad dentro del código HDL. De esta forma es posible conocer la forma en la que evolucionan las señales y determinar cómo es que estas afectan el diseño. El HDL Verifier provee interfaces de conexión con los simuladores HDL Cadence Incisive, Mentor Graphics ModelSim y Questa, los cuales son ejercitados mediante las entradas del algoritmo en Matlab o el modelo en Simulink.

3.2.2. Verificación del modelo de cosimulación mediante FPGA en el Lazo

El HDL Verifier automatiza la implementación de código HDL en tarjetas con FPGA, permitiendo así la verificación del diseño con FPGA en el lazo. Dicho modo de verificación complementa la cosimulación con simuladores HDL permitiendo de esta forma la prueba de diferentes escenarios de una forma más rápida. Además, este modo de verificación asegura que el algoritmo se comportará de la forma esperada en el mundo real. El HDL Verifier soporta la integración de un gran número de tarjetas compatibles con la verificación mediante FPGA en el lazo fabricadas por proveedores como Xilinx o Altera, a través de una interface de comunicación Gigabit Ethernet.

Capítulo 4

Implementación

Para iniciar con el desarrollo, se definieron los requerimientos que el diseño debe cumplir. Dichos requerimientos describen como se debe comportar el controlador a fin de llevar la planta al estado deseado y que componentes debe usar para lograr dicho objetivo. Los requerimientos para este diseño son los siguientes:

- El sistema podrá ser llevado a un punto de ajuste comprendido en el rango de -5 a 5 V_{DC}.
- La respuesta del sistema no deberá tener un sobrepaso mayor al 20 % con respecto al punto de ajuste.
- El tiempo de asentamiento deberá ser menor a 200 ms.
- El controlador debe ser capaz de actualizar su salida cada 500 μs.
- El controlador debe estar diseñado para trabajar con ADC y DAC bipolares de 12 bits.

4.1. Identificación de los componentes del sistema

Para definir el sistema de control se recurre a su representación básica mediante diagrama de bloques, tal y como se muestra en la Figura 4.1.

A continuación se describen las características de cada elemento.

4.1.1. Selección del punto de ajuste

En este bloque se encuentra la señal de ajuste, la cual representa el valor deseado al que se quiere llevar la salida de la planta. Este bloque cuenta con un generador de señales, el cual sirve para establecer una secuencia de puntos de ajuste, además cuenta con dos

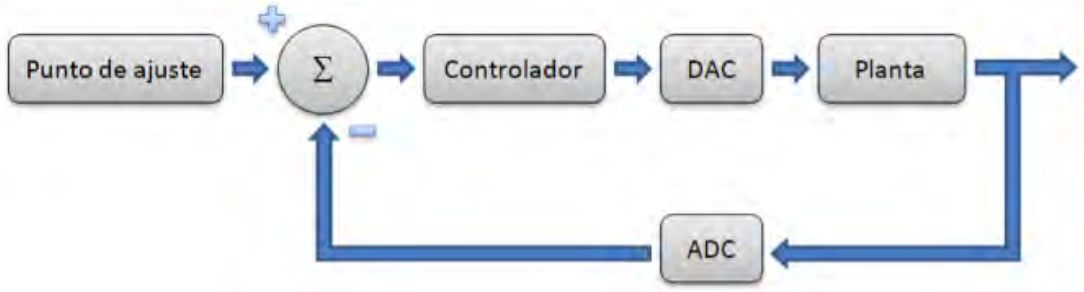

Figura 4.1: Representación básica de un sistema de control

señales, la señal que proporciona a la salida se denomina punto de ajuste analógico, la cual es limitada a un rango de -5 a 5 V_{DC}.

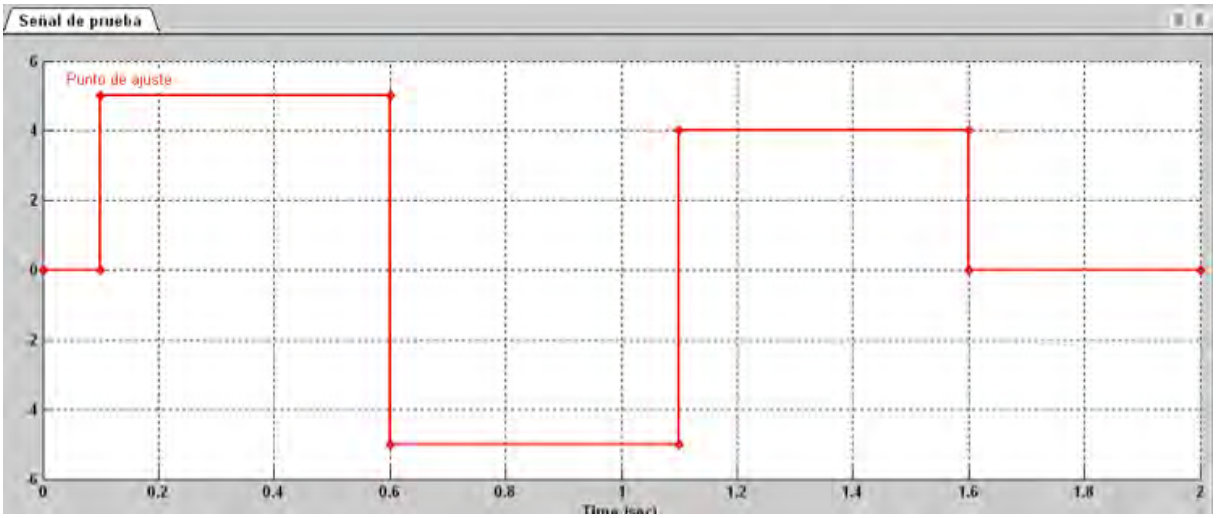

Figura 4.2: El Signal Builder, es usado para generar un patrón de señales de entrada

4.1.2. Convertidor ADC de punto de ajuste

Este bloque convierte la señal continua del punto de ajuste analógico proveniente del subsistema punto de ajuste en una señal digital, la cual puede ser procesada por el controlador digital. El bloque modela a un convertidor ADC con las características indicadas en la Tabla 4.1.

Cuenta con dos señales, la señal que proporciona a la salida se denomina punto de ajuste digital. Las señales tienes las siguientes características:

Parámetro	Valor
Bits	12
Resolución	4,096
Tiempo de conversión	500 μs
Tipo de entrada	Bipolar
Rango de entrada	[-8,7,996]
Tipo de dato de salida	Punto fijo, Signación
Codificación de salida	Complemento a 2

Tabla 4.1: Características del bloque de ADC punto de ajuste

4.1.3. Convertidor ADC de realimentación

Este bloque modela un convertidor analógico digital similar al convertidor analógico digital de punto de ajuste. Cuenta con dos señales, la señal que proporciona a la salida se denomina "variable digital controlada". Las señales tienen las características de la Tabla 4.2.

Señal	Tipo de dato	Rango	Precisión	Signo
Variable digital retenida	Doble precisión	[-5,5]	-	-
Variable digital controlada	fixdt(1,12,8)	[-8,7.99]	0.003902	Signado

Tabla 4.2: Características del bloque de ADC punto de ajuste

4.1.4. Bloque del controlador

Este bloque modela a la ley de control, la cual actuará dependiendo de las señales enviadas por los convertidores de punto de ajuste y realimentación. A partir de este bloque, el HDL Coder obtendrá una representación en VHDL y un archivo de programación, el que se utilizará para programar la tarjeta SP601. El subsistema proporciona una serie de señales, siendo la señal digital llamada "respuesta del controlador", la salida del controlador, la cual es la señal que indica el ajuste necesario que se debe realizar a fin de corregir el error. Las señales de este bloque tienen las características de la Tabla 4.3.

4.1.5. Convertidor DAC

Este bloque convierte la señal de ajuste generada en el control PI en una señal analógica capaz de interactuar directamente con la planta.

Señal	Tipo de dato	Rango	Precisión	Signo
error_12_7	fixdt(1,12,7)	[-16,15.992]	0.0078125	Signado
Respuesta del controlador	fixdt(1,12,7)	[-8,7.99]	0.0078125	Signado

Tabla 4.3: Características del bloque de ADC punto de ajuste

4.1.6. Bloque de la planta

Este bloque representa la dinámica del sistema a través de su función de transferencia continua. La entrada a este bloque proporciona la señal de ajuste necesaria para llevar al sistema al estado deseado, especificado mediante la señal de punto de ajuste, dicha señal está limitada por la ley de control al rango de -10 a 10 V_{DC}. Debido a que la planta deberá trabajar en el rango de los [-5,5] V_{DC}, considerando que el sobrepaso no deberá ser mayor de 20 %, el tipo de dato que describe señal de ajuste (respuesta del DAC) es un fixdt(1,12,7) y debido a que está limitada por la ley de control, solo tendrá valores en el rango de -10 a 10 V_{DC}. La salida de este bloque es una señal modelada por el tipo de dato de doble precisión, conocida como "variable analógica controlada". Dicha señal representa el estado actual de la planta.

4.2. Modelado del sistema mediante ecuaciones

Para este diseño en particular, se usa un control PI expresado en su forma paralela, en este caso, la salida es la suma de la ganancia proporcional K_p y la ganancia integral K_i. El control PI continuo en el dominio del tiempo en su forma paralela se representa mediante la ecuación

$$u(t) = K_p e(t) + K_i \int_0^\tau e(t)\ dt \tag{4.1}$$

Tomando la transformada de Laplace de esta ecuación se obtiene un control PI continuo en el dominio de la frecuencia en su forma paralela, representado por la siguiente función de transferencia

$$C(s) = K_p + \frac{K_i}{s}.$$

En Simulink, la función de trasferencia se representa utilizando identificadores distintos para la ganancia proporcional e integral.

$$C(s) = P + \frac{I}{s}. \tag{4.2}$$

donde

$$\begin{aligned} K_p &= P \\ K_i &= I \end{aligned}$$

Tomando la transformada Z de la función de transferencia en el domino de la frecuencia del control PI, obtenemos el controlador PI discreto en su forma paralela, el cual tiene la siguiente función de transferencia

$$C(z) = P + IT_s\frac{1}{z-1}.$$

donde T_s representa el tiempo de muestreo en segundos.
Una vez que se ha definido la estructura del controlador, el siguiente paso es calcular las ganancias de las acciones proporcional e integral. Simulink incluye un bloque que sirve para implementar controles PID llamado PID controller. Este bloque, permite implementar controles continuos o discretos. Además es posible ajustar las ganancias del control de forma manual o automática mediante un asistente de autoajuste por lo que se requiere el *S*imulink Control Design tool. La salida del bloque es una suma ponderada de la señal de entrada, la integral de la señal de entrada y la derivada de la señal de entrada. Los pesos de cada salida son la ganancia proporcional, ganancia integral y ganancia derivativa. Al seleccionar la opción de ajuste se muestra una aplicación interactiva, la cual permite observar la respuesta del sistema al aumentar o disminuir el tiempo de respuesta. Una vez que la forma de la salida tiene la forma deseada, los valores de las ganancias proporcional e integral aparecen en la sección de parámetros del controlador. Aunque se tiene que considerar que la respuesta es para una señal de entrada de escalón unitario, por lo que el sobrepaso y tiempo de establecimiento son diferentes para el cambio máximo en el punto de ajuste, es decir, para una entrada escalón de magnitud 10 V_{DC}.

Las ganancias del controlador para un tiempo de muestreo de $T_s = 500$ μs son las siguientes

$$P = 0.9982 \tag{4.3}$$
$$I = 43.924 \tag{4.4}$$

Mientras que las ganancias del controlador para un tiempo de muestreo $T_s = 5$ ms son

$$P = 0.955 \tag{4.5}$$
$$I = 44.27 \tag{4.6}$$

Finalmente podemos obtener la función de transferencia del controlador en forma discreta, la cual será de gran utilidad cuando se modele el controlador mediante el bloque

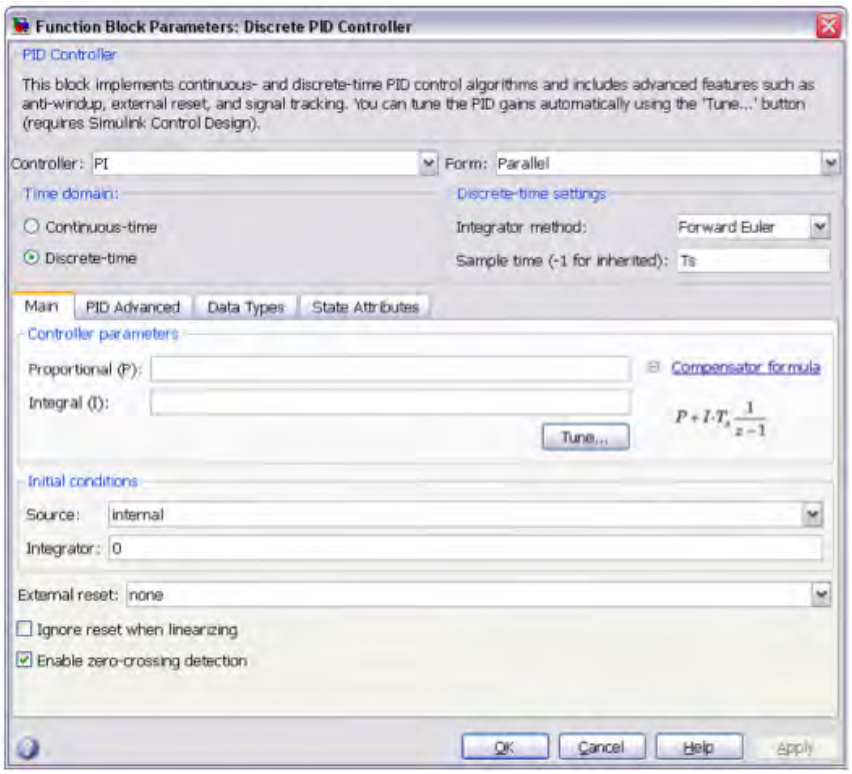

Figura 4.3: Uso del bloque Discrete PID Controller para el cálculo de ganancias

de función de transferencia discreta. Mediante Matlab, se definen las ganancias K_p y K_i, obtenidas en el asistente de autoajuste, así como el tiempo de muestreo. A partir de estos datos, se crea un objeto PI mediante el comando pid(Kp,Ki), lo que genera un control PI continuo en la forma paralela. Se obtiene una representación discreta en la forma paralela de controlador mediante el comando c2d(), finalmente se genera su representación en función de transferencia discreta mediante el comando tf() tal y como se muestra en la Figura 4.4.

Este código permite definir las funciones de transferencia del controlador en su forma paralela continua y discreta, así como el cociente de dos polinomios:

$$C(z) = \frac{0.998z - 0.9762}{z - 1} \tag{4.7}$$

en donde $T_s = 0.0005s$. Además también se define la función de transferencia de la planta en el domino de la frecuencia de la siguiente forma:

$$H(s) = \frac{e^6}{s^3 + 300s^2 + 3000s + e^6} \tag{4.8}$$

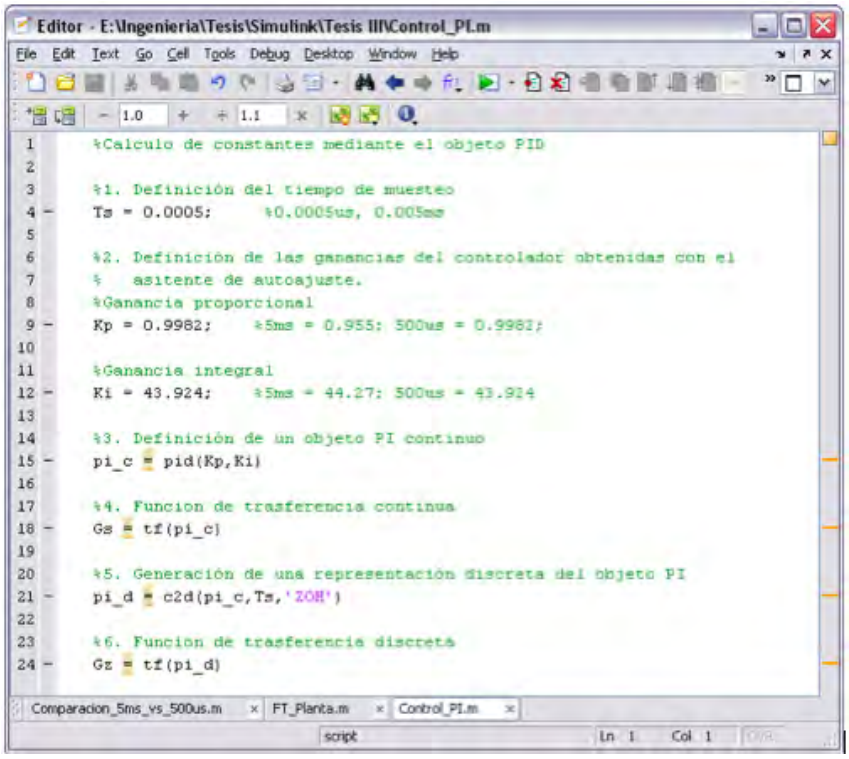

Figura 4.4: Generación de la función de transferencia discreta del controaldor

4.3. Construcción del diagrama de bloques

Una vez que se han identificado los componentes del sistema y se han definido las ecuaciones que los representan, el siguiente paso es conformar el modelo que permitirá la interacción de cada bloque, a fin de llevar la planta al estado deseado. Simulink permite explorar a través de diferentes formas de modelado del sistema. Sin embargo, es importante tener en cuenta que los bloques empleados deben ser compatibles con la generación de código HDL. Para esto, Simulink permite la generación de una librería llamada hdlsupported que se muestra en la Figura 4.5, la cual contiene elementos sintetizables únicamente. La utilización de elementos sintetizables únicamente es válida para el diseño bajo prueba, debido a que este será el elemento a partir del cual se generará el código VHDL. Para el resto del sistema, se puede usar cualquier bloque en la librería de Simulink. De acuerdo a los bloques identificados anteriormente, el siguiente paso es la creación de los diagramas de bloques correspondientes.

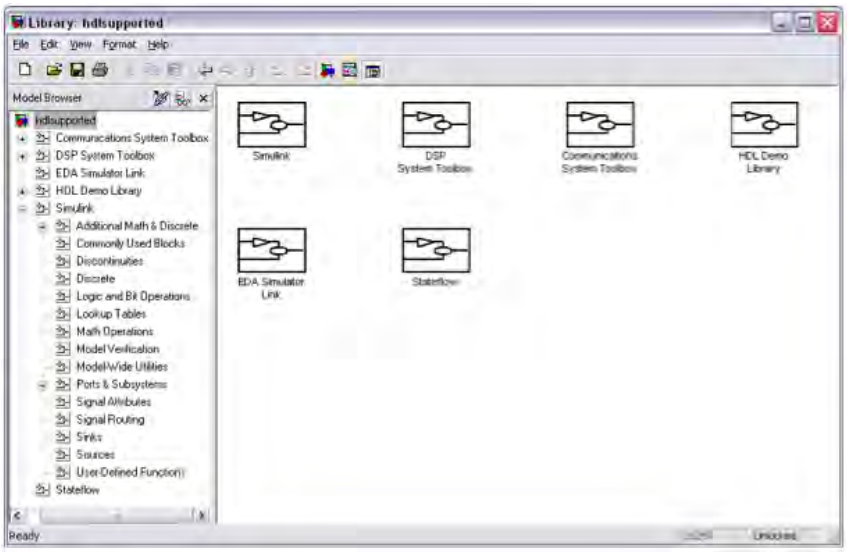

Figura 4.5: Librearía hdlsupported.

4.3.1. Subsistema de selección de punto de ajuste

Este subsistema es modelado mediante un generador de señales, el cual nos permite definir la forma de la salida. Dicha salida representa los puntos de ajuste a los que se desea llevar la planta. Así mismo, se cuenta con un bloque de limitación, que impide que el punto de ajuste salga del rango de -5 a 5 V_{DC}. El subsistema se muestra en la figura 6.7.

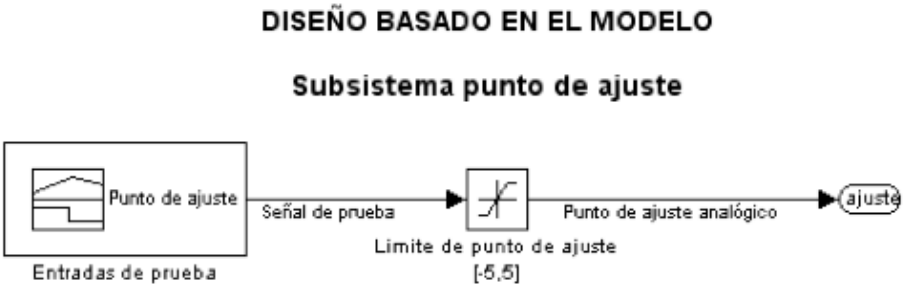

Figura 4.6: Bloque del subsistema punto de ajuste

4.3.2. Convertidor ADC de punto de ajuste

El subsistema emplea un reten de orden cero para emular la acción de muestreo y retención de la señal en el proceso de conversión analógico-digital, además se incorpora

un bloque que permite convertir señales, en este caso el bloque ADC realiza la conversión cada 500 μs. El subsistema se muestra en la Figura 4.7. El bloque de conversión incorpora

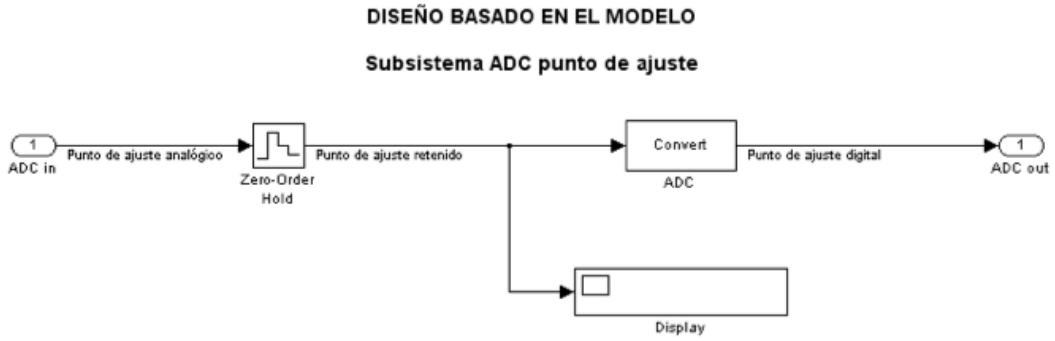

Figura 4.7: Bloque del subsistema conversión ADC de punto de ajuste

una interfaz gráfica para definir los rangos mínimo y máximo de salida, con el objetivo de encontrar el tipo de dato adecuado que sea capaz de representar dichas salidas. Las opciones que ofrece el bloque de conversión es que se puede seleccionar el tipo de dato.

4.3.3. Convertidor ADC de realimentación

El subsistema del convertidor de realimentación es modelado de la misma forma que el convertidor de punto de ajuste y se muestra en la Figura 4.8.

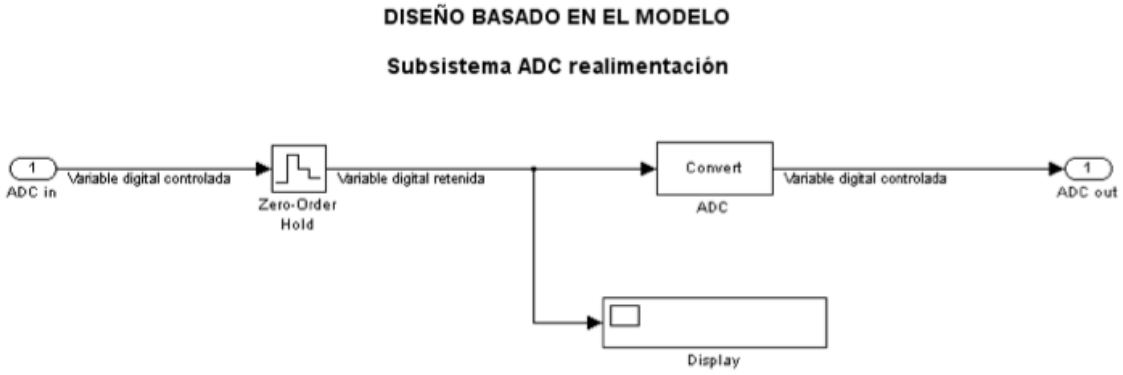

Figura 4.8: Bloque del subsistema conversión ADC de realimentación

4.3.4. Subsistema de controlador

Este subsistema encapsula la ley de control, la cual es posible describir de diferentes formas. En este proyecto se implementó el mismo tipo de control mediante tres formas distintas. Los tres subsistemas calculan el error de la misma forma, mediante el bloque de diferencia "b-error" se reciben las señales de punto de ajuste y de variable controlada en su versión digital. Mediante la diferencia de éstas se genera una señal que representa el error existente entre el valor deseado y el estado actual de la planta. También incluyen un bloque de limitación "límite de salida", este bloque asegura que la señal de control no sobrepasará el rango de -10 a 9.99 V_{DC}. Las tres formas de representar el control, se describen a continuación.

Modelado por bloque integrador

Una forma de modelar el control, es mediante los bloques de producto, adición, sustracción e integración como se indica en la Figura 4.9. De acuerdo a la forma paralela del controlador, esta ley de control se puede expresar como la suma de la parte proporcional, la cual se obtiene de multiplicar el error por la ganancia proporcional y la parte integral, la cual se obtiene directamente del bloque integrador. La acción PI se obtiene de la adición de las señales de salida de los bloques de producto e integración.

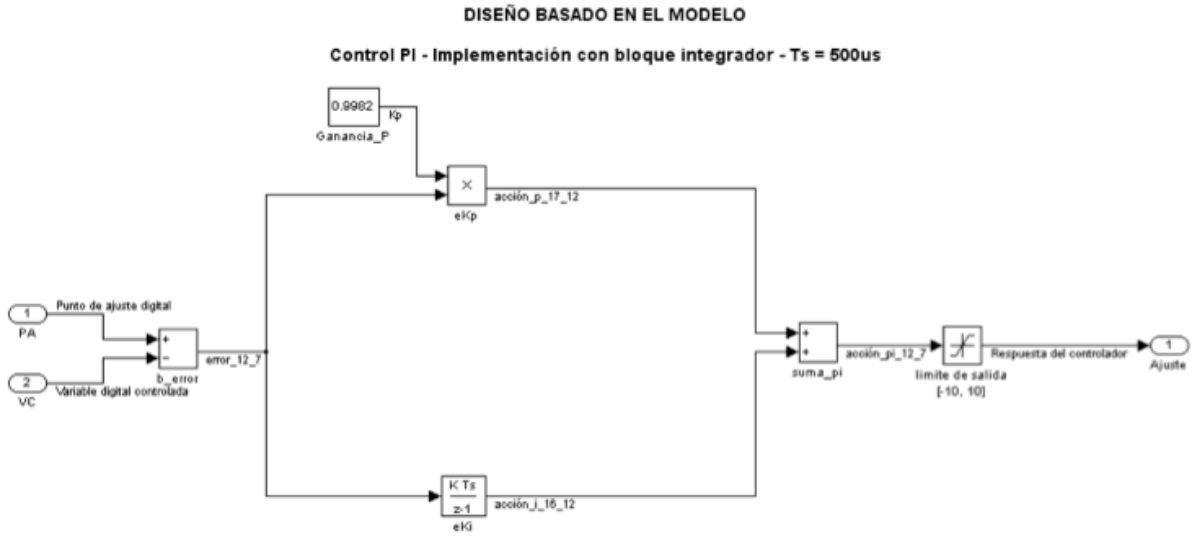

Figura 4.9: Modelado de la ley de control mediante bloque integrador

Modelado por bloques básicos

Una forma alterna de modelado consiste en sustituir el bloque integrador por bloques básicos, tales como bloques de ganancia, adición y retardo unitario. De la forma paralela del controlador discreto, la acción PI se obtiene como la suma de las acciones proporcional e integral. La parte proporcional se obtiene de multiplicar el error por la ganancia proporcional. Mientras que la parte integral se obtiene multiplicando el error por la ganancia integral, sumarlo a la salida anterior mediante el bloque de retardo unitario y multiplicarlo por el tiempo de muestreo. En este caso, la acción del retardo unitario es mantener un registro de la salida anterior, lo que permite ir acumulando el error. Esta representación permite tener una idea más clara de cómo se puede inferir el código HDL por parte del sintetizador, debido a que el bloque de retardo unitario se sintetiza directamente en un DFF (Flip Flop tipo D) con reset asíncrono y habilitación síncrona. El bloque que modela a la acción de control se muestra en la Figura 4.10.

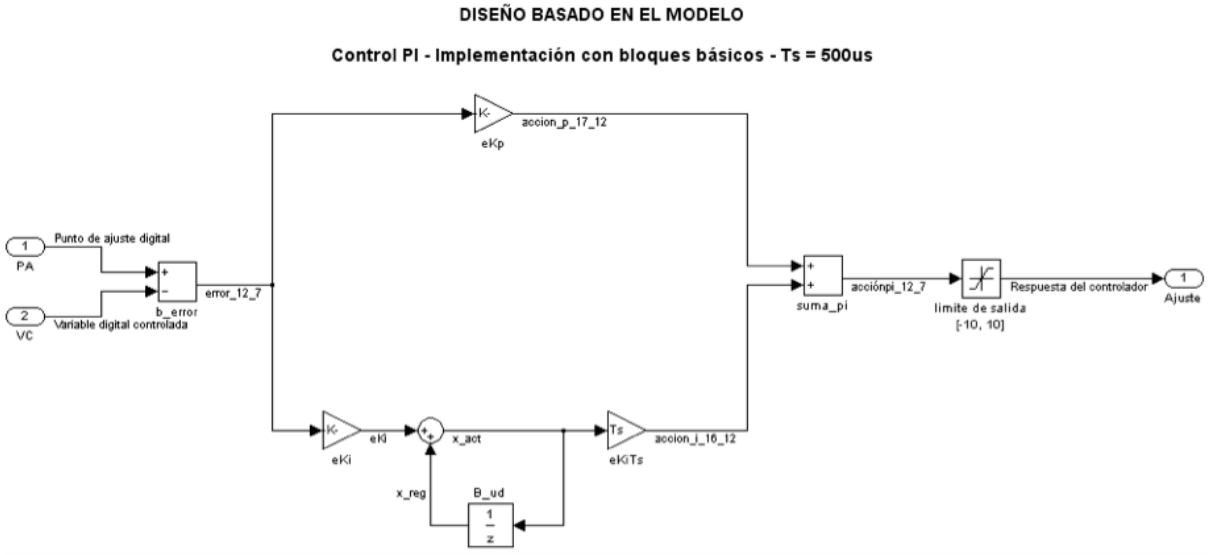

Figura 4.10: Modelado de la ley de control mediante bloques básicos

Modelado por bloque de función de transferencia

Al obtener la función de transferencia discreta del controlador, podemos expresar la ley de control como el cociente de dos polinomio e introducirlo en el bloque de función de transferencia discreta, lo cual simplifica el diagrama a bloques. Sin embargo, el modelado por función de trasferencia es una representación más abstracta, debido a que la relación entre el bloque de función de transferencia y los elementos sintetizables no es directa. Sin embargo el HDL Coder es capaz de reconocer la estructura de dicha función de transferencia e inferir la misma estructura que en los modelos anteriores. El modelado de la ley

de control mediante el bloque de función de transferencia discreta se muestra en la Figura 4.11.

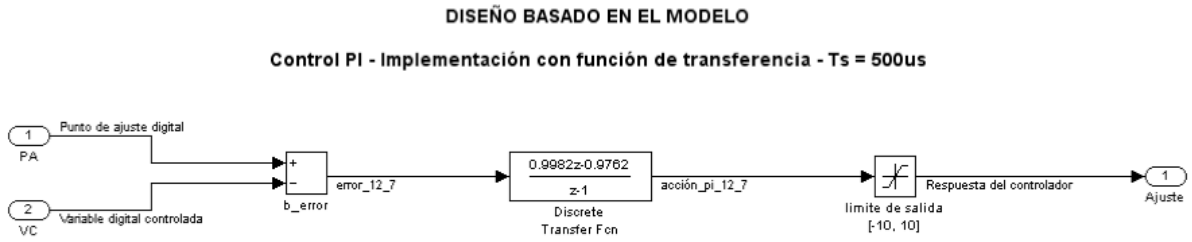

Figura 4.11: Modelado de la ley de control mediante el bloque de función de transferencia

Convertidor DAC

Este subsistema modela a un convertidor DAC de 12 bits bipolar, capaz de actualizar la salida cada 500 μs y se muestra en la Figura 4.12. Mediante el bloque DAC, la señal de punto fijo es convertida a una del tipo de doble precisión.

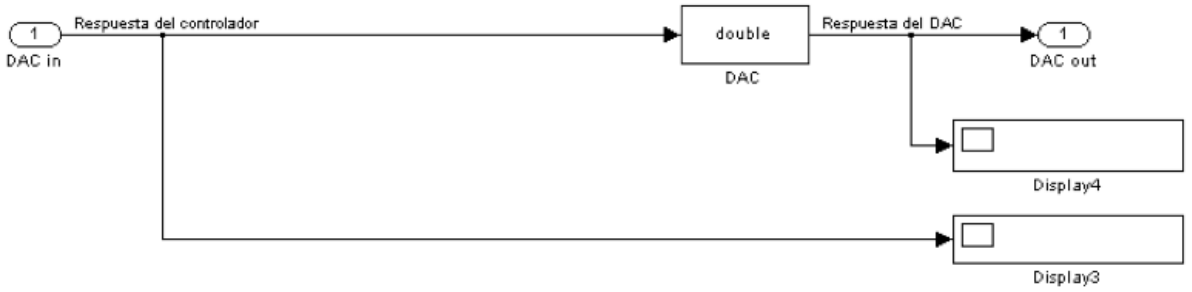

Figura 4.12: Bloque de convertidor DAC

4.3.5. Diagrama de bloques completo

Una vez definidos todos los subsistemas, se procede a la generación del modelo completo, interconectando cada subsistema de acuerdo al modelo del sistema básico de control mostrado en la Figura 4.1. Además se incluyen visualizadores y registradores de señales para poder observar el estado del sistema. El sistema completo se muestra en la Figura 4.13.

El punto central en este modelo lo constituye el subsistema llamado “control PI”, también conocido como DUT (diseño bajo prueba). El subsistema Control PI encapsula la ley de control y además, es la parte del modelo de donde se obtendrá el código HDL necesario para describir el FPGA.

4.4. Resultados de la simulación

Antes de pasar a la simulación, resulta conveniente realizar un ajuste en el tipo de dato empleado en el controlador. Para esto se cuenta con una herramienta llamada Fixed Point Tool, la cual realiza una simulación del sistema y captura los valores máximos y mínimos producidos. Al hacer esto, el Fixed Point Tool puede encontrar el tipo de dato más adecuado para representar las señales permitiendo así hacer un mejor use de los recursos del FPGA.
La simulación del sistema permite conocer con anticipación si el control tendrá un desempeño aceptable. Mediante la estimulación del sistema con cambios en la señal de valor deseado, se puede observar el comportamiento del sistema. En este caso se realizaron pruebas con diferentes puntos de ajuste. El modelo incluye un visualizador, el cual permite observar la respuesta de la planta y del control al cambiar el punto de ajuste. Al simular el sistema, se obtiene la salida que se muestra en la Figura 4.14, la cual muestra como el controlador es capaz de seguir los cambios en el punto de ajuste, sin exceder los límites establecidos en los requerimientos del control.

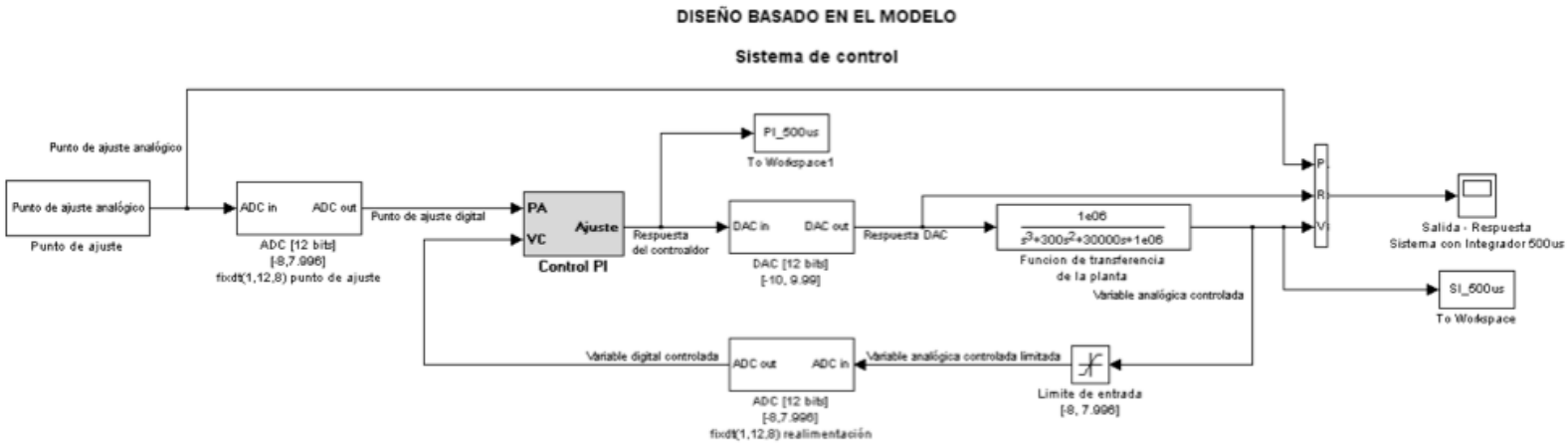

Figura 4.13: Diagrama completo del sistema de control

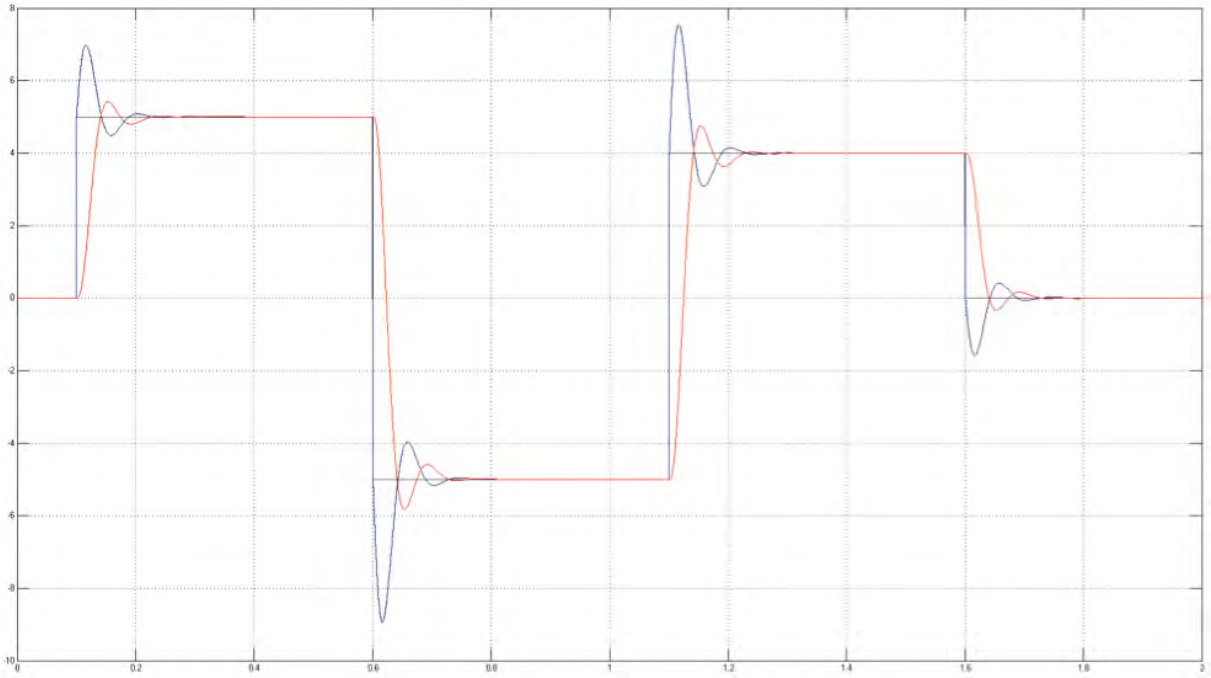

Figura 4.14: Respuesta del sistema con integrador con muestreo de 500 μs

Adicionalmente a los cambios en la señal de punto de ajuste, es posible experimentar con cambios en los parámetros internos del control. Mediante la modificación del tiempo de actualización del controlador es posible conocer la reacción del sistema a tiempos de actualización más largos, como se muestra en la Figura 4.15. Al simular el sistema con tiempo de actualización de 5 ms, se hace evidente que la repuesta del sistema no cumple las especificaciones requeridas, debido principalmente a que presenta un sobrepaso mayor al 20 % establecido.

4.5. Implementación

Una vez que se ha simulado el sistema y éste cumple con los requerimientos de diseõ, el siguiente paso es la verificación del modelo, para validar que el DUT pueda ser sintetizado. Mediante el Compatibility Checker, se puede saber si es posible generar código VHDL a partir del DUT. Si es posible dicha generación se notifica a través de su explorador web como se muestra en la Figura 4.16. Una vez que se ha comprobado la compatibilidad de generación de código VHDL, el siguiente paso es ejecutar el HDL Workflow Advisor, el cual guía al usuario a través de todo el proceso, desde la selección de la tarjeta a emplear en la verificación, la creación de modelos de cosimulación para FIL y simuladores HDL, generación de reportes hasta la programación de la tarjeta SP601.
Una vez que se ha generado el código HDL, independientemente de si se generaron modelos

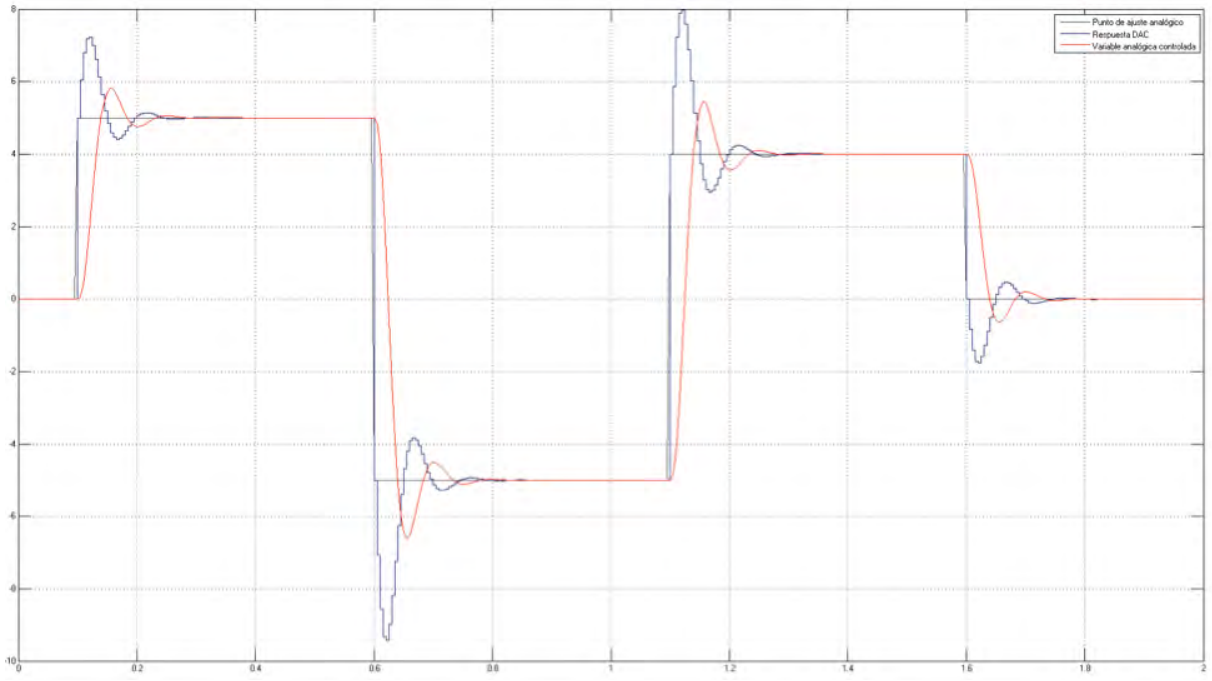

Figura 4.15: Respuesta del sistema con integrador con muestreo de 5 ms

de cosimulación, el HDL Workflow Advisor arroja reportes acerca de la implementación, como se muestra en la Figura 4.17. La primera parte del reporte es un resumen del proyecto, el cual detalla las características del mismo.

4.5.1. Recursos consumidos

Al habilitar la opción de generación de reporte de utilización de recursos, el HDL Coder, genera un reporte que resume la cantidad y tipo de recursos empleados en el diseño, los cuales pueden ser multiplicadores, sumadores/restadores, registros, RAMs y multiplexores. Así mismo, genera los hipervínculos necesarios para identificar que bloques hacen uso de estos recursos. Este reporte es especialmente útil cuando se requiere una optimización en el diseño, al comparar la cantidad y tipo de recursos, es posible obtener mejores dependiendo de los requerimientos del sistema.
Es importante tener en cuenta que los recursos consumidos mostrados en este reporte son aquellos relacionados con el DUT únicamente. Es decir, no muestra los recursos empleados para establecer el protocolo de comunicación con la tarjeta SP601. Sin embargo, es posible conocer un estimado del total de recursos utilizados en el FPGA. Cuando el diseõ se va a verificar mediante la técnica de FPGA en el lazo, el HDL Workflow Advisor crea una carpeta donde almacena todos los archivos necesarios para programar el FPGA. Dicha carpeta incluye un archivo de tipo proyecto, el cual se ejecuta con el Xilinx ISE Web Pack.

Figura 4.16: Reporte de compatibilidad de generación de código VHDL.

Al abrir el archivo y ejecutar el proceso de síntesis, arrojará un resumen con la cantidad estimada de recursos empleados. Por otro lado, al ejecutar el proceso de implementación, el Xilinx ISE Web Pack, genera un reporte con la cantidad exacta de recursos consumidos.

4.5.2. Trazabilidad

El HDL Coder tiene la capacidad de crear reportes de trazabilidad, la cual es una característica importante en sistemas de generación de código automático. La trazabilidad permite establecer una relación entre los bloques del modelo en Simulink y el código VHDL generado, lo que permite conocer como se sintetiza cada parte del DUT. El reporte de trazabilidad lista todos los bloques contenidos en el modelo en Simulink y mediante hipervínculos, es posible identificar cada componente dentro del modelo. Debido a que la generación de código VHDL es obtenida únicamente del DUT, muchos componentes del modelo no son trazables. Sin embargo, la sección de interés está en el subsistema Control PI. Bajo el concepto de trazabilidad se distinguen dos tipos:

- La trazabilidad modelo a código establece un hipervínculo entre los bloques de interés en Simulink y el código VHDL generado, de esta forma es posible conocer qué parte del código en VHDL fue generada por cada bloque del modelo en Simulink.

- La trazabilidad código a modelo, permite identificar el bloque que generó cada parte del códifo. Teniendo de esta manera una trazabilidad bidireccional entre el modelo y el código VHDL generado.

4.5.3. Prueba y verificación

Una vez que se ha generado el código HDL, el paso final es la prueba y verificación del diseño. El HDL Coder y HDL Verifier permiten la verificación del mediante modelos

Figura 4.17: Reporte generado por el HDL workflow advisor

de cosimulación basados en simuladores HDL y mediante la integración de FPGAs en el lazo.

Verificación del modelo de cosimulación mediante el simulador HDL ModelSim

El HDL Coder permite la generación de modelos de cosimulación para trabajar con simuladores HDL específicos, en este proyecto, se genera un modelo de cosimulación para trabajar con el simulador HDL desarrollado por Mentor Graphics: ModelSim PE Student Edition 10.1. Al habilitar la opción de generación de modelo de cosimulación para ModelSim, automáticamente se genera una copia del modelo original tal y como se muestra en la Figura 4.18.

El nuevo modelo incluye un nuevo bloque llamando bloque de cosimulación además de algunos subsistemas auxiliares. El bloque de cosimulación permite establecer una interfaz de comunicación entre el simulador HDL y Simulink, los bloques se muestran en la Figura 4.19.

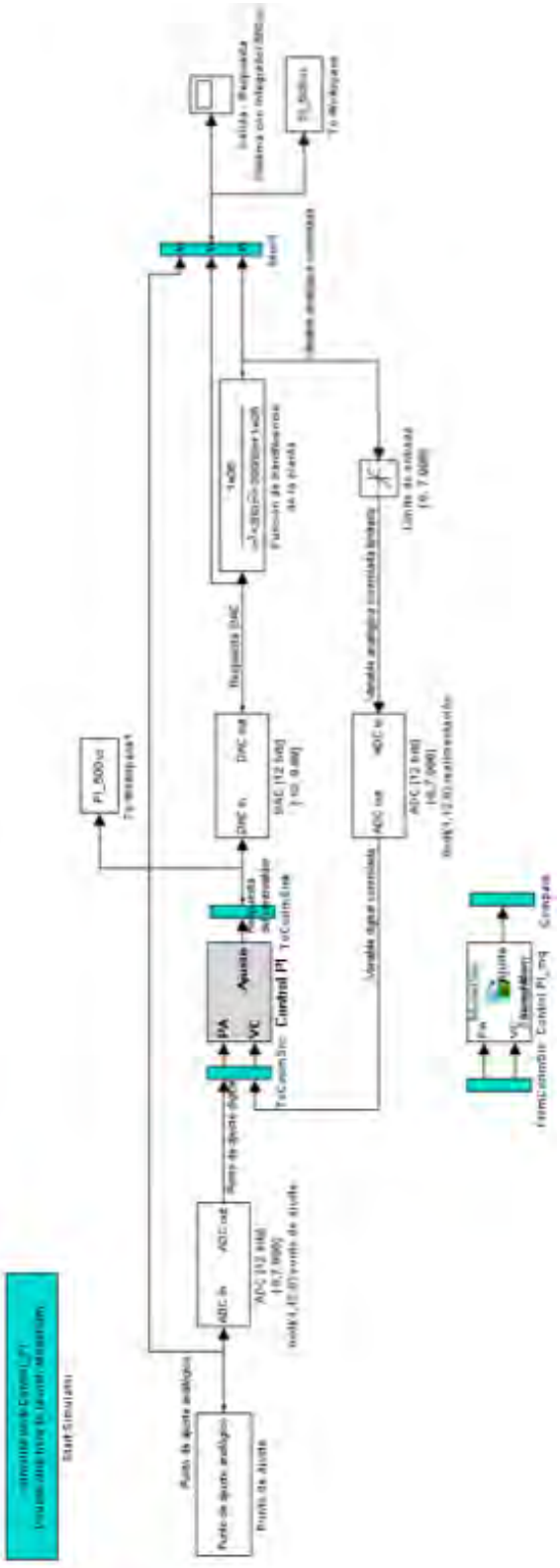

Figura 4.18: Modelo de cosimulacion para simulador HDL

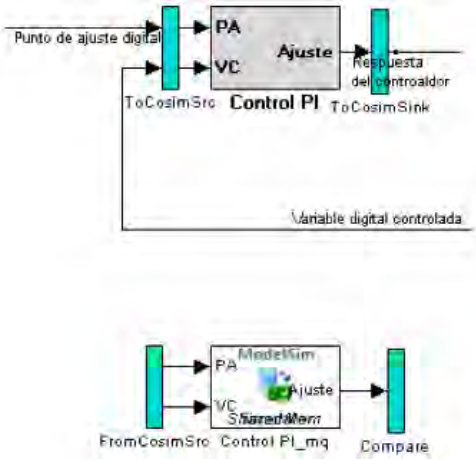

Figura 4.19: Bloques de cosimulacion

Cuando la interacción se da entre el simulador HDL y un modelo en Simulink, el simulador HDL funciona como servidor y Simulink como cliente. En este caso, el simulador HDL responde a la petición de simulación que recibe desde el bloque de cosimulación en el modelo en Simulink. El modo de comunicación entre ambos simuladores depende de la forma en como se ejecuten las aplicaciones, es decir, si los simuladores se están ejecutando en el mismo equipo, el modo de comunicación basado en "memoria compartida" es el modo adecuado, debido a que presenta el rendimiento óptimo. De otra forma, si las aplicaciones se encuentran dentro de un red, la opción de comunicación será a través de sockets TCP/IP. En este diseño, ambos simuladores se están ejecutando en el mismo equipo, por lo que se selecciona el modo de comunicación por memoria compartida. En el modelo de cosimulación generado, las señales de entrada y salida del diseño bajo prueba, en este caso el control PI, son interceptadas por dos subsistemas llamados "ToCosimSrc" y "ToCosimSink". El objetivo del subsistema "ToCosimSrc" es capturar los estímulos que originalmente actúan sobre diseño bajo prueba y trasmitirlos al bloque de cosimulación, el cual se encarga de transmitirlos al simulador HDL. El subsistema ToCosimSink únicamente pone disponible la respuesta del diseño bajo prueba para ser comparada. Por otra parte, la respuesta del simulador HDL es recibida en Simulink mediante el bloque de cosimulación. La salida es enviada al subsistema "Compare". En este subsistema la salida del diseño bajo prueba y la salida del comparador son comparadas. Realizando la sustracción de ambas señales es posible conocer si existe un error entre la repuesta del sistema simulado mediante Simulink el simulador HDL. Las respuestas de ambos simuladores así como la diferencia entre ambas señales se puede observar en la Figura 4.20.

Además de observar las señales en Simulink, también se pueden observar en ModelSim. Una característica importante del simulador HDL es que permite la visualización de señales tanto en su representación digital como analógica, lo que puede resultar útil al momento de identificar fallas en el diseño.

4.5.4. Verificación del modelo de cosimulación mediante FIL (FPGA in the Loop)

El HDL Coder también permite la verificación del diseño mediante la integración de hardware en el lazo de control. El HDL Coder puede generar un modelo de cosimulación específicamente diseñado para trabajar con una tarjeta compatible con FPGA en el lazo. En este diseño se emplea la tarjeta de prueba Xilinx Spartan 6 SP601. Al ejecutar el último paso del HDL Workflow Advisor, automáticamente se genera una copia del modelo original llamado modelo de cosimulación FIL. El nuevo modelo incluye un nuevo bloque llamando bloque de cosimulación FIL además de algunos subsistemas auxiliares. Este bloque de cosimulación FIL sirve como interface entre la tarjeta SP601 y el modelo en Simulink, dicho de otra forma, este bloque nos permite integrar el hardware al lazo de control con el fin de el FPGA pueda interactuar con el resto del modelo como si se tratara de cualquier otro bloque. El bloque de cosimulación FIL provee un canal de comunicación

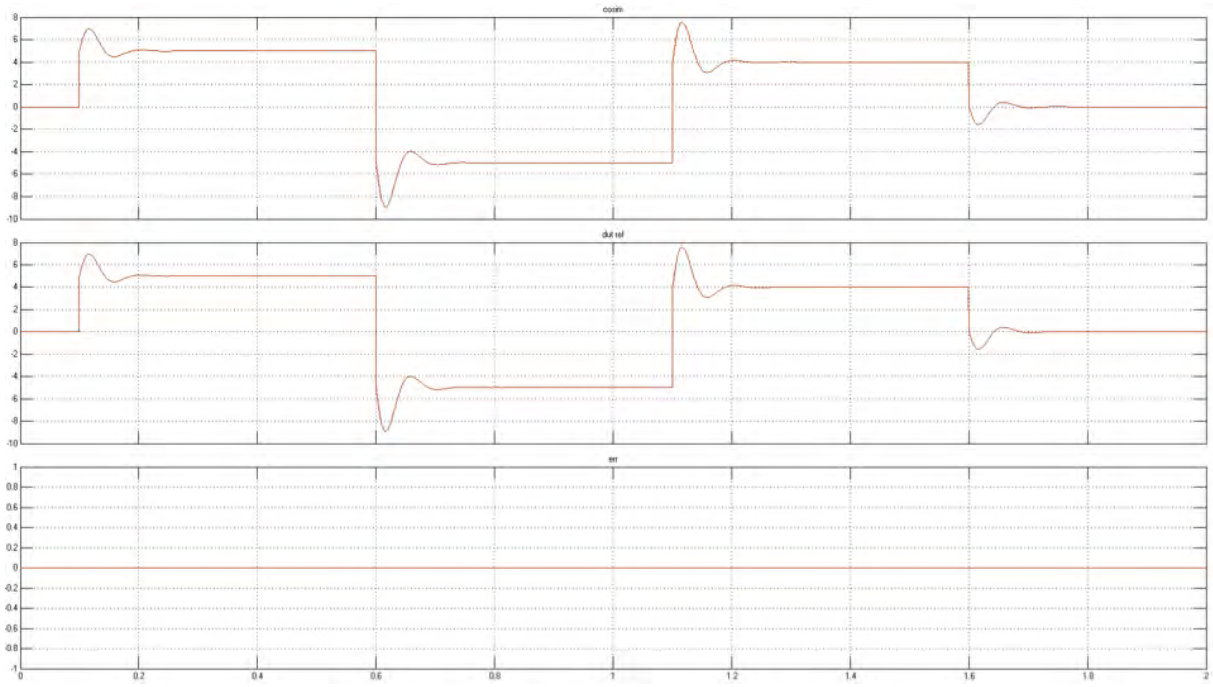

Figura 4.20: Comparación de las respuesta del sistema

para enviar y recibir información entre Simulink y la tarjeta SP601. Este canal usa una conexión Gigabit Ethernet. Debido a que la comunicación entre Simulink y la tarjeta SP601 es estrictamente sincronizada, la simulación FIL provee un método de verificación más confiable. El sistema completo pude observarse en la Figura 4.21. Una vez descargado el archivo de programación, simulamos el sistema y comparamos las salidas del bloque PI original denominado DUT_ref (diseño bajo prueba) y el bloque de cosimulación FIL, el cual representa el mismo control siendo ejecutado en el FPGA. La respuesta de la tarjeta es recibida en Simulink mediante el bloque de cosimulación FIL, posteriormente la salida es enviada al subsistema "Compare". En este subsistema la salida del diseño bajo prueba y la salida de la Tarjeta SP601 son comparadas, realizando una substracción de ambas señales, es posible determinar si existe un error en la cosimulación.
El bloque de cosimulación FIL proporciona automáticamente un visulaizador, el cual permite comparar la salida de cada bloque y observar el error que pudiera existir. En este caso, la salida de los controles DUT_ref (diseño bajo prueba) y FIL es el mismo, por lo que no existe error, como se muestra en la Figura 4.22.

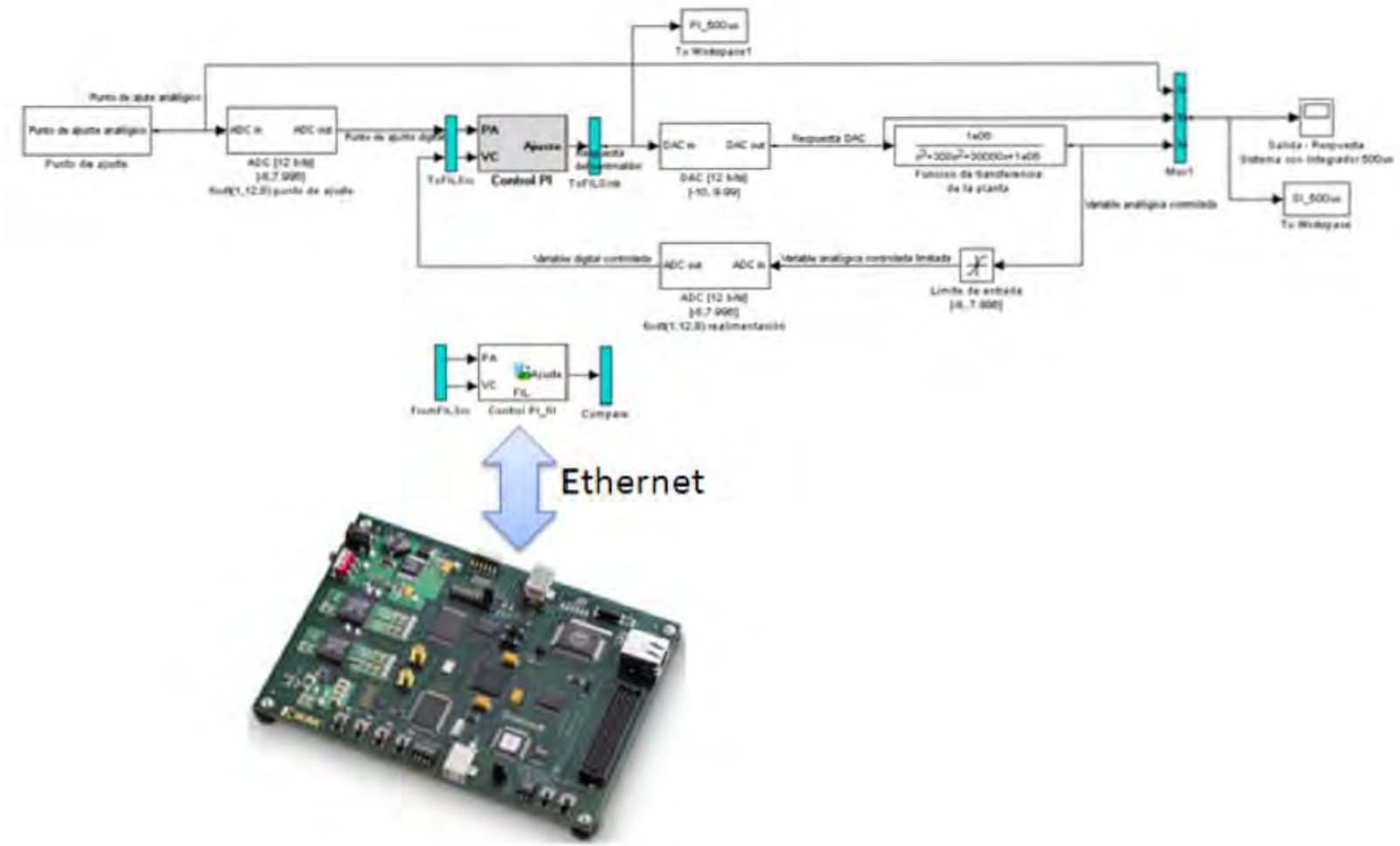

Figura 4.21: Interacción entre el modelo en Simulink y la tarjeta SP601

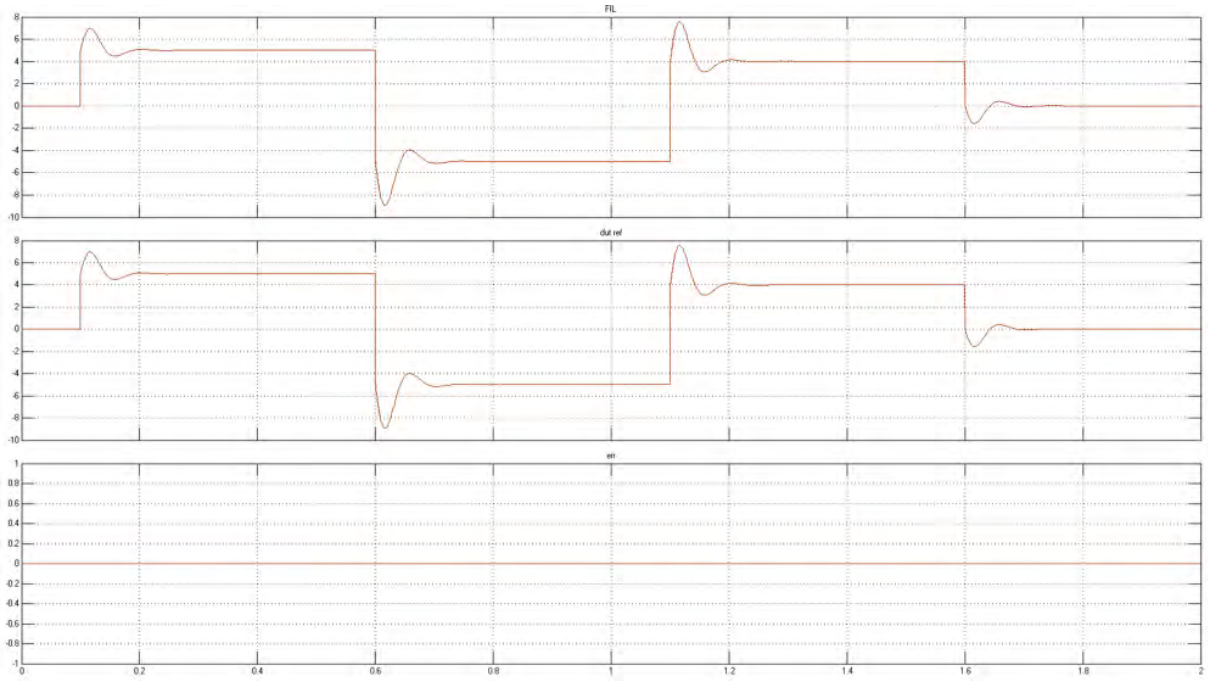

Figura 4.22: Comparación de las respuestas del modelo de cosimulación para FIL

Capítulo 5

Conclusiones

En el presente trabajo se introduce el concepto de verificación de sistemas mediante técnicas de cosimulación y hardware en el lazo, el cual se presenta como una valiosa herramienta de diseño de sistemas de control. Ya que debido a la creciente complejidad de los nuevos sistemas de control, es necesario realizar pruebas más exhaustivas tanto al circuito electrónico como al algoritmo de control que determina su comportamiento. Las técnicas de cosimulación permiten ir más adentro en el sistema y conocer la evolución de las señales en todo momento. Debido a esta característica el tiempo de diseño se reduce y se puede generar un algoritmo más estable. La técnica de hardware en el lazo permite validar el algoritmo y saber de forma anticipada si el algoritmo de control cumple con los requerimientos establecidos. Estas técnicas constituyen sólo una etapa dentro del flujo de diseño de sistemas electrónicos de control. Sin embargo, resultan de gran utilidad ya que siempre es deseable conocer posibles vulnerabilidades del sistema antes de que el producto final salga al mercado.

En este trabajo de tesis se simuló e implementó un controlador PI usando técnicas de verificación y hardware en el lazo, comprobando la efectividad de las herramientas tanto de software como de hardware. Dicha efectividad radica en gran medida en la versatilidad de conexión entre los programas de simulación y el hardware además de la facilidad que implica la utilización de la la tarjeta Spartan 6, convirtiendo complejas instrucciones de lenguaje VHDL en un sistema de control totalmente funcional.

En cuanto a los conceptos de control automático, se pudo comprobar que las herramientas utilizadas son idóneas para implementar sistemas de control de manera efectiva. Se realizó a lo largo del trabajo un listado de cada uno de los bloques de un sistema de control clásico en lazo cerrado, aproximándolos al plano de la implementación y comprobando su funcionalidad.

5.1. Trabajo futuro

Dentro del trabajo a futuro que se pudo identificar a lo largo esta tesis se encuentra:

- Probar la metodología estudiada con plantas físicas simulándolas previamente y comparando los resultados.
- Elaborar un manual de usuario para que la metodología establecida sea usada de forma sistemática tanto en plantas digitales y como en físicas.

Bibliografía

[1] Bergeron Janick. *Writing Testbenches: Functional Verification of HDL Models*. Kluwer Academic Publishers. New York, 2002.

[2] Roth Charles. *Digital Systems Design using VHDL*. PWS Publishing Company. Boston MA, 1998.

[3] Garibay Martínez Alberto Ramiro. *Control en tiempo real de procesos dinámicos empleando módulos de tecnología FPGA*. Trabajo de investigación UAM Azcapotzalco, 2007.

[4] Viswanathan V. Sornam. *Embedded Control Using FPGA*. Seminar Report. Indian Institute of Tecnlology. Bombay, 2005.

[5] Chu Pong. *FPGA Prototyping by VHDL Examples*. John Wiley & Sons Inc. New Jersey, 2008.

[6] Ogata Katsuhiko. *Ingeniería de Control Moderna*. Pearson Eduacación. Madrid, 2003.

[7] The MathWorks Inc. *HDL Coder User's Guide*. 2012.

[8] The MathWorks Inc. *HDL Verifier User's Guide*. 2012.

[9] Chang K. C. *Digital Systems Design with VHDL and Synthesis*. IEEE Computer Society. California, 1999.

[10] Fulks Charles. *Model Based Design for FPGA Development*. Embedded Systems Conference. Boston, 2009.

[11] Downey Allen B. *Physical Modeling in Matlab*. Class Notes. Needham MA, 2007.

[12] Chu Pong P. *RTL Hardware Design using VHDL*. John Wiley & Sons Inc. New Jesey, 2006.

[13] The MathWorks Inc. *Simulink Control Design User's Guide*. 2012.

[14] The MathWorks Inc. *Simulink Fixed Point User's Guide*. 2012.

[15] The MathWorks Inc. *Filter Design HDL Coder User's Guide*. 2012.

[16] The MathWorks Inc. *HDL Coder User's Guide*. 2012.

[17] Xilinx. *Spartan 6 FPGA Data Sheet*. 2010.

[18] Ashenden Peter y Lewis Jim. *The Designer's Guide to VHDL Third Edition*. Systems on Silicon. California, 2008.

[19] Perry Douglas L. *VHDL Programming by Example*. McGraw Hill. California, 2003.