UNIVERSIDAD NACIONAL DE INGENIERÍA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO DE UNA PLATAFORMA DE DESARROLLO BASADA EN UN FPGA ORIENTADA A SISTEMAS DE ADQUISICIÓN REMOTA EN TIEMPO REAL SOBRE UNA RED ETHERNET

TESIS

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

FÉLIX ALBERTO PURILLA FLORES

PROMOCIÓN

2010 - II

LIMA – PERÚ
2011

DISEÑO DE UNA PLATAFORMA DE DESARROLLO BASADA EN UN FPGA ORIENTADA A SISTEMAS DE ADQUISICIÓN REMOTA EN TIEMPO REAL SOBRE UNA RED ETHERNET

A mi padre, por sus experiencias transmitidas,
A mi tía Elena, por haberme abierto las puertas de su hogar,
A mi hermana Jessica, por hacerme ver fáciles las cosas difíciles,
A mi hermana Kathya, por enseñarme que siempre
hay algo bueno que sacar de las cosas malas,
Y finalmente con todo mi cariño a María Graciela,
a quien nunca tendré forma de agradecerle
su amor infinito.

SUMARIO

El presente trabajo está enmarcado dentro de la línea de investigación de sistemas de control en tiempo real. El prototipo desarrollado, el cual está basado en un kit de desarrollo FPGA Spartan 3E y Java Tiempo Real, constituye una herramienta para el control de la velocidad angular del eje de un motor DC en tiempo real desde una computadora con sistema operativo Linux vía protocolo Ethernet. La implementación de este sistema tiene como finalidad servir de módulo educativo para que sea usado en los laboratorios de los cursos relacionados a sistemas de control y sistemas tiempo real.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

PRÓLOGO

En cursos de pregrado y posgrado donde se enseñan la teoría de control moderno y avanzada respectivamente es necesario contar con plataformas de desarrollo que permitan a los estudiantes aplicar la teoría aprendida.

Una plataforma de desarrollo que contenga un convertidor DC-DC basado en tecnología BJT, un FPGA que recoja información del sensor y maneje al convertidor y un motor DC con encoder que sirva como planta a controlar sería de utilidad para probar distintos algoritmos de control en un computador que pueda ejecutar tareas en tiempo real.

En este trabajo se muestra una plataforma, basada en el kit de desarrollo Spartan 3E, la cual contiene los siguientes bloques principales:

a) Comunicación con la PC. La comunicación con la PC se realiza a través del puerto Ethernet lo que facilita el intercambio de datos entre el FPGA y la computadora en tiempo real.
b) Adquisición de señales. Para lo cual se ha desarrollado una IP[1] que permite la lectura del encoder de cuadratura de un motor DC.
c) Generación de señales de control. Para lo cual se ha desarrollado una IP que genera dos señales PWM para el control del motor DC.

Así también incluye, un convertidor DC-DC de cuatro cuadrantes basado en tecnología BJT y un acondicionador de señales para la correcta lectura del encoder.

Como parte final del prototipo se muestra la interface de usuario desarrollada en Java (RTSJ para la implementación del algoritmo de control y Java para la representación gráfica de la salida controlada).

El primer capítulo, está dedicado al estado del arte para este tipo de sistemas, aplicaciones y las ventajas de usar un FPGA.

El segundo capítulo, incluye el marco teórico necesario para entender el diseño e implementación de la plataforma de desarrollo.

---

[1] Intelectual Property

El tercer capítulo, contiene el diseño de la plataforma bajo los lineamientos requeridos: bajo precio, comunicación con la computadora en tiempo real y el manejo de herramientas de software libre.

El cuarto capítulo, constituye la concepción del prototipo en el FPGA: implementación de las IP en el FPGA, programación de procesador MicroBlaze, programación en RTSJ, programación JAVA y desarrollo de las interfaces electrónicas.

El quinto capítulo, presenta el proceso realizado para obtener el modelo de un motor DC, así como, la implementación de dos sistemas de control de velocidad del motor DC y un estudio sobre los efectos negativos en el control causado por la pérdida de paquetes en la red.

Las siguientes líneas son de agradecimiento a las personas e instituciones que han apoyado este trabajo

En primer lugar mi agradecimiento al Msc. Ing. Eleazar Sal y Rosas por los consejos sobre el camino que debería tomar la tesis y por la dedicación que puso desde el principio hasta el final de éste.

Así también agradezco al Instituto de Investigación de la FIEE, bajo la dirección de la Ing. Judith Betteta, por los equipos y el laboratorio proporcionado sin el cual este trabajo no podría haberse terminado.

Finalmente, con todo mi cariño, el agradecimiento a todos mis amigos y familia en especial para Elena, Félix, Jessica, Kathya y María Graciela.

CAPÍTULO I
ESTADO DEL ARTE

1.1. Sistemas de control en red en tiempo real

El auge de las redes de comunicación ha introducido el concepto de sistema controlado en red, también conocido como NCS por sus siglas en ingles (Networked Control System) [1].

La definición clásica de un NCS es como sigue: Cuando un sistema de retroalimentación de control es cerrado a través de un canal de comunicación, el cual, puede ser compartido con otros nodos del sistema, como es el caso de la red Ethernet, este sistema de control es llamado NCS. Un NCS también se puede definir como un sistema de control por retroalimentación en el cual los lazos de control se cierran a través de una red.

La principal característica de un NCS es que la información (referencia de entrada, salida de la planta, la entrada de control, etc.), puede ser intercambiada entre los nodos (planta, sensores, controlador, actuador) de la red de comunicación [1].

1.1.1. Ventajas y Aplicaciones de los sistemas de control en red.

Desde hace muchos años, las tecnologías de redes de datos están siendo aplicadas dentro del sector industrial y militar. Estas aplicaciones incluyen las plantas de fabricación, automóviles, aviones, redes ferroviarias, redes satelitales, etc. La conexión de los componentes de un sistema de control para estas aplicaciones a través de una red, tiene como objetivo una mejor distribución de los componentes (en el caso que estos lo requieran). Un ejemplo de ello son los sistemas de control de un vehículo, en el cual los sensores y actuadores se comunican con el controlador a través de una red CAN. Partiendo de este principio, la conexión de la internet con el sistema que se desea controlar, es actualmente uno de los grandes temas de investigación a nivel mundial. Tales sistemas de control por la internet están siendo utilizadas en: exploraciones espaciales, exploración terrestre, automatización de fabricas, diagnostico y solución de problemas a distancia en

entornos peligrosos o no, robots domésticos, automóviles, aviación, monitoreo de plantas de fabricación, tele robótica y tele operación.

1.1.2. Categorías de un NCS

En términos generales, los dos tipos principales de sistemas de control que utilizan las redes de comunicación son los sistemas de control en red compartida (Estructura directa) y sistemas de control remoto (Estructura jerárquica) cuyas definiciones pueden ser encontradas de manera más amplia en la referencia [1].

El uso compartido de los recursos de una red para la transferencia de las mediciones, de los sensores a los controladores y las señales de control provenientes de los controladores hacia los actuadores, puede reducir enormemente la complejidad de las conexiones.

En la figura 1.1 se muestra la primera configuración de un NCS, lo cual proporciona una mayor flexibilidad en la instalación, además de facilitar el mantenimiento y solución de los problemas. El intercambio de datos entre los nodos de control permite una mejor distribución del hardware utilizado para este propósito, es decir para el caso de las lecturas de los sensores se necesitaría un microcontrolador de bajas prestaciones en comparación con el controlador central, el cual debe de tener por ejemplo: un sistema operativo embebido, un multiplicador en punto flotante, etc.

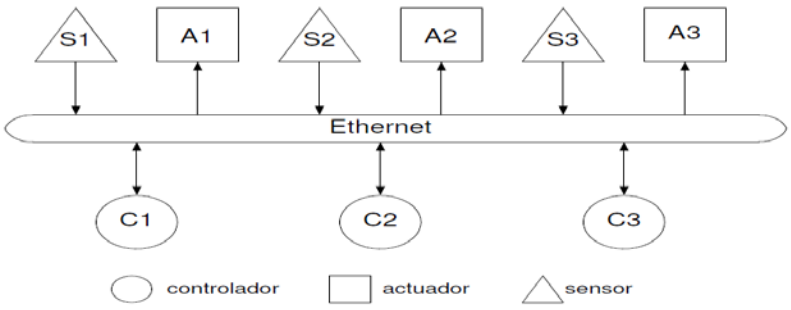


Fig.1.1 Estructura directa de un NCS (Fuente: Elaboración propia)

Por otro lado, un sistema de control remoto puede ser diseñado de una manera distribuida que incluya que grandes distancias. En la figura 1.2 se representa este tipo de sistemas.

El lugar donde un controlador central se instala normalmente se denomina un "sitio local", mientras que el lugar donde se encuentra la planta se denomina sitio remoto, así pues el CM se encuentra en el sitio local y en el lado remoto se encuentran los nodos C1, C2 y C3.

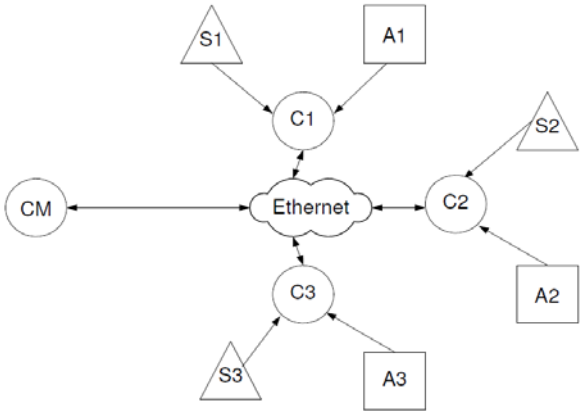


Fig.1.2 Estructura jerárquica de un NCS (Fuente: Elaboración propia)

Por otro lado, las aplicaciones de control en red pueden ser divididas en dos categorías: (1) aplicaciones sensibles al tiempo de retardo y (2) aplicaciones insensibles al retraso de tiempo. En aplicaciones sensibles al retraso de tiempo, el tiempo de respuesta es crítico, es decir, si el retraso de tiempo supera el límite especificado de tiempo tolerable, la planta o el dispositivo puede ser dañado o producir un rendimiento inferior.

Un ejemplo de aplicaciones sensibles al retraso de tiempo es la tele-operación a través de redes para las operaciones en una planta nuclear.

De otro lado, las aplicaciones insensibles al tiempo de retardo son las tareas o programas que se ejecutan en tiempo real, pero cuyos plazos no son críticos. Ejemplos de estas aplicaciones son el correo electrónico, FTP, DNS, y HTTP.

1.1.3. Componentes de un NCS

Sin importar cuál sea la disposición o modalidades utilizadas para conectar y configurar el hardware y software de un sistema de control en red, los componentes utilizados tienen que permitir cuatro funciones que constituyen la base funcional de una NCS, estos son: adquisición de la información (sensores), procesamiento de la información (controlador), intercambio de la información (comunicación) y actuación sobre la planta (actuadores).

a) Adquisición de la información

Es la parte del sistema que recoge la información de todos los sensores, el cual no necesariamente debe disponer de conexión directa a una red.

b) Procesamiento de la información

Este procesamiento es realizado por el controlador del sistema en el caso de una estructura directa o los controladores en el caso de una estructura jerárquica (control distribuido), este controlador puede estar basado en un PLC, DSP, o un computador de propósito general.

c) Intercambio de la información

Según las necesidades de escalabilidad, simplicidad y costos entre otros son distintos las configuraciones y tipos de red que se pueden escoger, tenemos el protocolo CAN, MODBUS, UDP/IP, RTP ó TCP/IP entre otros.

d) Actuación sobre la planta

Esta función es realizada por los elementos finales de control, los cuales pueden o no disponer de conexión directa a una red.

1.2. Estado del arte en sistemas de control embebido

Un sistema embebido (embedded system) es un sistema electrónico diseñado para realizar una o algunas funciones específicas dentro de un sistema más grande al que ayuda a gobernar, éstas funciones pueden implicar tareas como, el procesamiento de la información generada por sensores o el control de determinados actuadores.

Por lo general un sistema embebido (SSEE) está basado en uno o más procesadores digitales los cuales pueden presentarse como:

- Microprocesadores
- Microcontroladores
- Procesador digital de señales
- Dispositivos FPGA

El software que maneje estos dispositivos tendrá requisitos específicos que dependerán del tipo de aplicación, sin embargo se puede mencionar las características en común que tendrá un software para un SSEE:

- Operación en tiempo real
- Optimización del uso de los recursos disponibles

En la actualidad son dos las alternativas principales que nos permiten desarrollar sistemas operativos en tiempo real sobre un sistema operativo utilizando herramientas de software libre y que disponen de productos electrónicos para implementar sistemas de

control, de medición y comunicación: Java Tiempo Real (RTSJ) y Linux Tiempo Real (RTAI). Empresas dedicadas a este rubro son, por ejemplo, Ajile system, que comercializa sistemas basados en Java Tiempo Real y Comedi que comercializa RTOS Linux embebidos.

1.2.1. Java Tiempo Real Embebido

Ajile System es una empresa que desarrolla y vende microprocesadores que usan un kernel RTOS aJile basado en una plataforma Java. Los sistemas embebidos de aJile son utilizados de manera masiva en routers, dispositivos de control y monitoreo inteligentes, siendo el Chip Aj102, el controlador automático JNIB-102 y el kit de inicio AJ102SK los productos más utilizados en los sistemas de control. Mas información respecto de estos productos puede encontrarse en la referencia [2].

El System on Chip (SoC) Aj102 es el segundo dispositivo en una familia de productos system-on-chip, éstos, ejecutan directamente instrucciones en bytecode de la JVM, además de hilos primitivos de Java tiempo real. Este chip fue sacado al mercado en noviembre del 2009, lo que demuestra que la tendencia actual está orientada a sistemas de control tiempo real.

Este procesador, trabaja directamente en Java (bytecodes), mediante el uso de su máquina virtual, lo que elimina la necesidad de algún intérprete y provee el rendimiento óptimo de Java tanto en requerimiento de memoria como en tiempo de ejecución (aproximadamente 1 us en tiempo de ejecución por instrucción).

La programación de sistemas Tiempo Real en Java está mucho más orientada a sistemas robóticos, por las mismas características del lenguaje (programación orientada a objetos), siendo este software a diferencia de J2SE, orientado a sistemas embebidos que aseguren alta performance, sincronización de objetos, planificación de tareas, manejo de entradas y salidas e interrupciones (mediante acceso directo a direcciones). El Aj102 es idealmente adecuado para routers de celulares 3G, PC’s industriales y sensores inteligentes conectados a una red.

Entre las características del procesador aJ102, podemos mencionar:

- Procesador Java de ejecución directa de 32 bits
- Periférico de controlador de interrupciones.
- Tres módulos de 24 bit para procesos de cuenta y temporización (timer/counter)
- Seis moduladores por ancho de pulso

- Temporización de perro guardián (Watchdog timer)

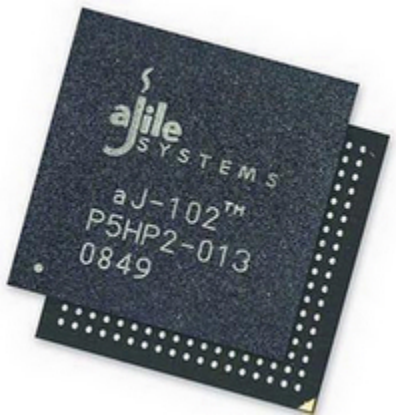


Fig.1.3 Procesador SoC aJ102

Por otro lado, la misma empresa comercializa módulos industriales, como el caso del JNIB-102 el cual, es un controlador automático compacto, programable en lenguaje Java, de bajo costo y con internet habilitado y ha sido diseñado para monitoreo en tiempo real, adquisición de datos y aplicaciones de control. Es idealmente adecuado para proveer acceso a la red a sensores y actuadores de baja potencia que requieran respuestas en tiempo real.

Las aplicaciones típicas incluyen automatización en casa (domótica), oficina, edificios, manejo y control de energía, monitoreo de tráfico, vigilancia y sistemas de seguridad.

Fig.1.4 Java Network Interface Box JNIB-102

Para el desarrollo de aplicaciones de usuario a nivel de desarrollo, aJile proporciona tres tipos de arquitectura SOC diferentes: aJ-100, aJ102 y aJ200.

El AJ102SK es un kit de inicio que posee al procesador aJ102 mencionado anteriormente; entre los periféricos más importantes que vienen en este kit tenemos:

- SDRAM de 32 Mb y Nand Flash de 32 Mb
- Puerto Ethernet 10/100 base T
- Puerto Serial

- Puerto anfitrión USB 2.0
- Zócalo para módulo ZigBee.

1.2.2. Linux Tiempo Real Embebido

Con respecto al proyecto Comedi, por sus siglas Control and Measurement Device Interface, este desarrolla drivers open-source, herramientas y librerías para adquisición de datos. Mayor información sobre estos productos puede encontrarse en la referencia [3].

ComediLib es una librería que provee una interface amigable de desarrollo para dispositivos Comedi. Incluido en la distribución ComediLib está la documentación, configuración y programas de demostración.

Kcomedilib es un módulo de un kernel Linux (distribuido con Comedi) que provee la misma interface como ComediLib en el espacio del kernel, este kernel es adecuado para tareas en tiempo real. Entre sus características tenemos:

- Soporte integrado tiempo real para casi todo hardware
- Librería de alto nivel (ComediLib)

1.3. Aplicación de sistemas embebidos en tiempo real

Un sistema en tiempo real es requerido cuando es necesario que las respuestas del sistema sean predecibles en cuanto a un plazo de tiempo para su ejecución.

Entre los principales sistemas que requieren procesos en tiempo real tenemos:

- Sistemas de control por computador
- Sistemas de tratamiento de señales
- Sistemas de mando y control
- Sistemas de telecomunicaciones
- Sistemas de aviónica
- Sistemas de control de trenes y autos
- Sistemas de control de tráfico aéreo

1.4. FPGA y sus aplicaciones en Tiempo Real

La principal ventaja de un FPGA es que su arquitectura interna es reconfigurable, esto debido a que se encuentra constituido por pequeños módulos (bloque básico de un FPGA) que según como sean configurados y conectados con otros pueden construir diversidad de circuitos y sistemas digitales.

En comparación con un microcontrolador, el cual tiene una arquitectura fija con un procesador y un determinado número fijo de periféricos, por ejemplo 3 moduladores por ancho de pulso, un FPGA puede implementar un procesador y una cantidad de moduladores por ancho de pulsos que está sólo limitada por la cantidad de bloques básico en el FPGA y la cantidad pines de salida que disponga el FPGA.

Esto implica que si por ejemplo en el microcontrolador anterior se quiere implementar otros tres moduladores por ancho de pulso esto tendría que hacerse por software, agregar hardware para la generación de PWM o en el peor de los casos cambiar el microcontrolador lo que involucraría un rediseño de la tarjeta.

Sin embargo un FPGA no tendría ese problema, ya que una vez implementado el procesador y sus tres generadores de PWM sólo tendría que unirse otros bloques básicos del FPGA que no estén siendo usados y así hacer los tres generadores de PWM más que están siendo requeridos.

Es por esta razón que un FPGA, es de por si totalmente versátil y es usado en sistemas donde se requiere alta velocidad de procesamiento y/o donde se prevé un gran flujo de datos de entrada y salida [4], por lo que es posible encontrar estos dispositivos en:

- Sistemas de alarma
- Climatización de autobuses
- Control industrial
- Electrónica espacial
- Electrónica submarina
- Equipos de medicina y radiología
- Identificación dactilar
- Navegación GPS
- Regulación de trafico
- Simuladores de vuelo
- Tarifación telefónica
- Telemedicina
- Sistemas de radar

1.5. Proyectos de investigación relativos a la tesis

Algunos de los institutos de conocidos laboratorios de investigación y que trabajan en NCS se enumeran a continuación [1].

- Advanced Diagnosis, Automation and Control (ADAC) Laboratory en la Universidad del estado de Carolina del Norte (http://www.adac.ncsu.edu/).
- Alleyne Research Group en la Universidad de Illinois en Urbana-Champaign (http://mr-roboto.me.uiuc.edu/).
- Networked Control Systems Laboratory en la Universidad de Washington (Seattle) (http://www.ee.washington.edu/research/ncs/index.html).
- Center for Networked Communicating Control Systems (CNCS) en la Universidad de Maryland en College Park (http://www.isr.umd.edu/CNCS/).
- Network Control Systems Laboratory en la Universidad Nacional de Taiwán (http://cc.ee.ntu.edu.tw/ncslab/).
- Interdisciplinary Studies of Intelligent Systems en la Universidad de Notre Dame (http://www.nd.edu/isis/).

1.5.1. Control remoto basado en un microcontrolador y Ethernet embebido

Este proyecto realizado por Imran Ahmed, Hong Wong, y Vikram Kapila en la Universidad Politécnica de Brooklyn tiene como finalidad el control experimental vía Ethernet de un motor DC [5].

En este proyecto se logró comunicar el microcontrolador BS2P40 con una página web cliente remota alojada en una PC. La posición del motor DC se obtuvo con un potenciómetro acoplado al eje del motor además de usar un tacómetro para medir la velocidad, estas señales luego de ser digitalizadas se envían mediante datagramas UDP al algoritmo de control en la PC remota, mientras que el comando del ángulo y la ganancia de control son enviados desde la PC remota al microcontrolador. Entre sus principales logros tenemos:

- La implementación de un controlador PD sobre el microcontrolador BS2P40 para controlar el motor DC.
- La implementación de un applet de Java sobre una página web cliente en la PC remota para graficar los datos recibidos por el sensor, esto es, la posición del motor.

1.5.2. Implementación de una red MODBUS/TCP

En este proyecto se desarrolla e implementa una red de instrumentación y control industrial con conectividad TCP/IP (como por ejemplo Internet), capaz de ser supervisada y controlada remotamente a través del protocolo Modbus / TCP usando el sistema

embebido TINI de Dallas Semiconductor. Además se desarrolla una interfaz de usuario gráfica para acceso desde Internet vía Web [6]. Entre sus principales logros tenemos:

- La implementación de un servidor MODBUS / TCP en el SSEE TINI usando Java.
- La implementación de una interfaz web (Java) para el monitoreo de datos que llegan al SSEE TINI.

1.5.3. Implementación de una RTU en un FPGA

En este proyecto se presentó el diseño de una RTU (Unidad Terminal Remota) empleando herramientas de descripción de hardware para el diseño de los módulos y herramientas de System-on-Chip para la integración [7].

La implementación se lleva a cabo en una tarjeta de desarrollo de sistemas embebidos de la familia de FPGA Stratix II de Altera. Se hace uso de un sistema operativo en tiempo real MicroC/Os encargado de administrar los recursos de hardware/ software de la RTU, a través de una interfaz HTML, empleando TCP/IP se accede a la interfaz de diagnóstico, la comunicación entre la RTU y la estación Maestra se da con el protocolo MODBUS. Adicional a esto se diseño una interfaz I/O que permita la entrada o salidas de señales análogas y digitales provenientes desde un sensor o hacia algún actuador.

Entre sus principales logros se puede mencionar:

- Implementación de un módulo IP core para el control de un chip ADC 0808
- Implementación de un módulo IP core para el control de un chip DAC 0808
- Implementación de un módulo IP core para la generación de señales PWM
- Comunicación entre una PC y un FPGA haciendo uso de una conexión MODBUS/TCP
- Comunicación entre una PC y un FPGA haciendo uso de una conexión TCP/IP

CAPÍTULO II
MARCO TEÓRICO

2.1. Visión Global del Sistema

El objetivo principal de este trabajo es desarrollar un sistema que permita controlar remotamente un motor DC, de manera que una PC conectada a una red Ethernet reciba la velocidad del motor (salida de la planta) y luego esta, envíe un comando que represente la señal de control.

En referencia a la figura 2.1, se observa que de un lado se tiene una PC que enviará y recibirá datos, y del otro lado se debe tener un dispositivo que por lo menos permita comunicarse con la PC remotamente. Para este fin existen múltiples formas de abordar la solución, una de ellas podría ser el uso de un microcontrolador que no sólo serviría como interfaz para la comunicación Ethernet sino también, para la adquisición de velocidad y generación de PWM.

Sin embargo, un microcontrolador tiene una arquitectura fija, es decir, tiene un determinado número de dispositivos electrónicos que no pueden ser modificados; por otro lado un FPGA, al no tener una arquitectura fija, permite la escalabilidad y versatilidad del sistema, por lo cual, en el caso que se requiera aumentar el número de periféricos PWM sólo se tendría que desarrollar otra IP en VHDL, lo cual no sería posible en un microcontrolador dado que su arquitectura es fija.

El FPGA entonces, deberá permitir la comunicación con la PC remota, procesar la señal del encoder para obtener la velocidad angular del eje del motor, previo acondicionamiento de señales, y generar señales de PWM para el circuito de potencia.

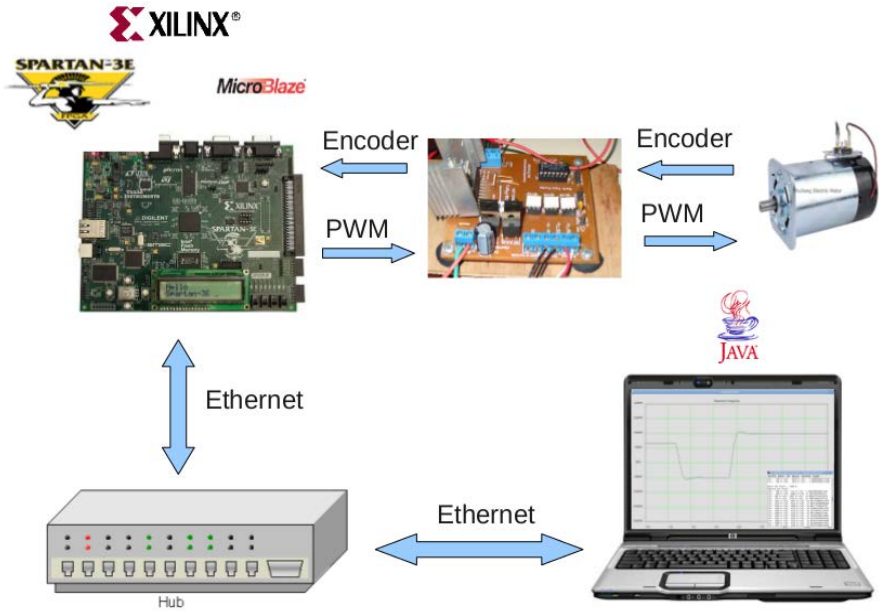


Fig.2.1 Visión Global del Sistema (Fuente: Elaboración propia)

Entonces, basándonos en la figura 2.1, se puede adelantar que periódicamente (intervalo de muestreo) el FPGA procesará y enviará la velocidad del eje del motor a la PC la cual hará el cálculo de la señal de control para luego enviársela al FPGA, finalmente se cerrará el bucle con la recepción y el establecimiento del comando PWM el cual es realizado por el FPGA, lo cual debe suceder antes del comienzo del próximo intervalo de muestreo.

2.2. Field Programmable Gate Arrays (FPGA)

Desde la aparición del Circuito Integrado existen dos alternativas para la solución de problemas usando sistemas digitales. La primera y la más difundida consiste en la codificación de un algoritmo en un **microprocesador**, es decir, usando un dispositivo que tiene una arquitectura fija (microcontrolador, microprocesador, DSP) se programa una secuencia de instrucciones en el microprocesador para que este resuelva la tarea.

La segunda alternativa llamada **ASIC** (Application Specific Integrated Circuit) apareció en la década de los 80’s y consiste en el mapeo directo del algoritmo en el hardware, es decir, se diseña el sistema digital que resolverá el problema y este se construye en un chip de silicio.

La mayoría de problemas se resuelven usando sistemas digitales basados en microprocesadores, sin embargo cuando la entrada y salida de datos combina gran cantidad

y velocidad, o cuando el número de operaciones por muestra es demasiado elevado la única opción es la realización de un ASIC.

Un circuito integrado de aplicación especifica es un chip diseñado completamente a medida, por lo que el parámetro Área-Tiempo-Potencia es mínimo y por tanto un ASIC resulta ser óptimo, además que un ASIC ofrece alta confidencialidad, elevada frecuencia de operación y mayor densidad de transistores por milímetro cuadrado.

Sin embargo, a pesar de las ventajas citadas la solución de problemas basado en un enfoque ASIC no es siempre la mejor alternativa ya que estos son: bastante caros de producir, complejos en cuanto al diseño y además no se tiene la posibilidad de corregir errores una vez mandado a fabricar. Por estas razones la tecnología ASIC se usa sólo cuando no hay otra alternativa y para elevados volúmenes de producción.

Por otro lado los fabricantes de ASIC para acelerar el proceso de diseño y de producción introdujeron otras soluciones llamadas *Gates Arrays* y *Standard Cells*.

En la tecnología *Gates Arrays* se estandarizan las etapas iniciales de fabricación que tenían como producto final (de esta primera etapa) un arreglo de filas y columnas de transistores sin conectar. La conexión se realizaba en las últimas etapas de fabricación diseñando a medida las capas de metalizaciones finalizando así el proceso.

La tecnología *Standard Cells* en cambio no era un proceso pre manufacturado si no que se disponía de células prediseñadas y estandarizadas como RAM’s, ALU’s Multiplicadores, etc.

El siguiente paso fue dado gracias a los esfuerzos de los fabricantes de ASIC’s, los cuales se centraron en ofrecer un producto o una metodología de diseño que combinara el notable ATP de los ASIC con las ventajas de los microprocesadores en cuanto:

- Al bajo costo
- Alta estandarización
- Facilidad de corrección de errores y
- Reducción de tiempo de salida al mercado.

Es así que apareció la idea de un ASIC reprogramable, es decir, se dispuso reemplazar la interconexión fija de los Gate Arrays y Standard Cells por pistas metálicas conectables a través de algún medio.

Siempre teniendo en mente las ventajas y desventajas de los ASIC, Xilinx sacó al mercado un “Gate Array” programable por campo, había nacido el primer FPGA.

A diferencia de un “Gate Array” en el que su interconexión es fija, en un FPGA las interconexiones se sustituyen por una serie de pistas metálicas conectables por transistores de paso controlados por un conjunto de bits de control (bitstream) almacenados en una memoria interna.

2.2.1. El FPGA Spartan®-3E

La familia de FPGA’s Spartan®-3E consiste de 5 miembros que ofrecen densidades de sistemas de compuertas desde 100,000 a 1.6 millones tal como se muestra en la tabla [8].

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM bits(1)	Block RAM bits(1)	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S100E	100K	2,160	22	16	240	960	15K	72K	4	2	108	40
XC3S250E	250K	5,508	34	26	612	2,448	38K	216K	12	4	172	68
XC3S500E	500K	10,476	46	34	1,164	4,656	73K	360K	20	4	232	92
XC3S1200E	1200K	19,512	60	46	2,168	8,672	136K	504K	28	8	304	124
XC3S1600E	1600K	33,192	76	58	3,688	14,752	231K	648K	36	8	376	156

Notes:
1. By convention, one Kb is equivalent to 1,024 bits.

Fig.2.2 Composición de los FPGA de la familia Spartan3E (Fuente: Guía de usuario de la familia Spartan 3 [8] pág. 33)

La arquitectura de la familia Spartan®-3E consiste fundamentalmente de un conjunto de 5 elementos funcionales programables.

Bloques Lógicos Configurables (CLB’s), que contienen Look-Up Tables (LUT’s) que implementan circuitos lógicos además de contener elementos de almacenamiento como Flip-Flop.

Bloques de Input/Output (IOB’s), estos bloques controlan el flujo de datos entre los pines de entrada y/o salida y la lógica interna del dispositivo. Cada IOB soporta flujos bidireccionales de datos así como de lógica de tres estados.

Bloques de RAM (BRAM), que proveen una alternativa al almacenamiento de datos usando como base bloques de puertos duales de 18Kbit.

Bloques multiplicadores, que realizan la multiplicación de dos números binarios de 18 bits.

Bloques administradores de reloj digital DCM, que poseen auto calibración y soluciones totalmente digitales para el disminuir el retraso y efectuar división, multiplicación y desplazamiento de fase en las señales de reloj.

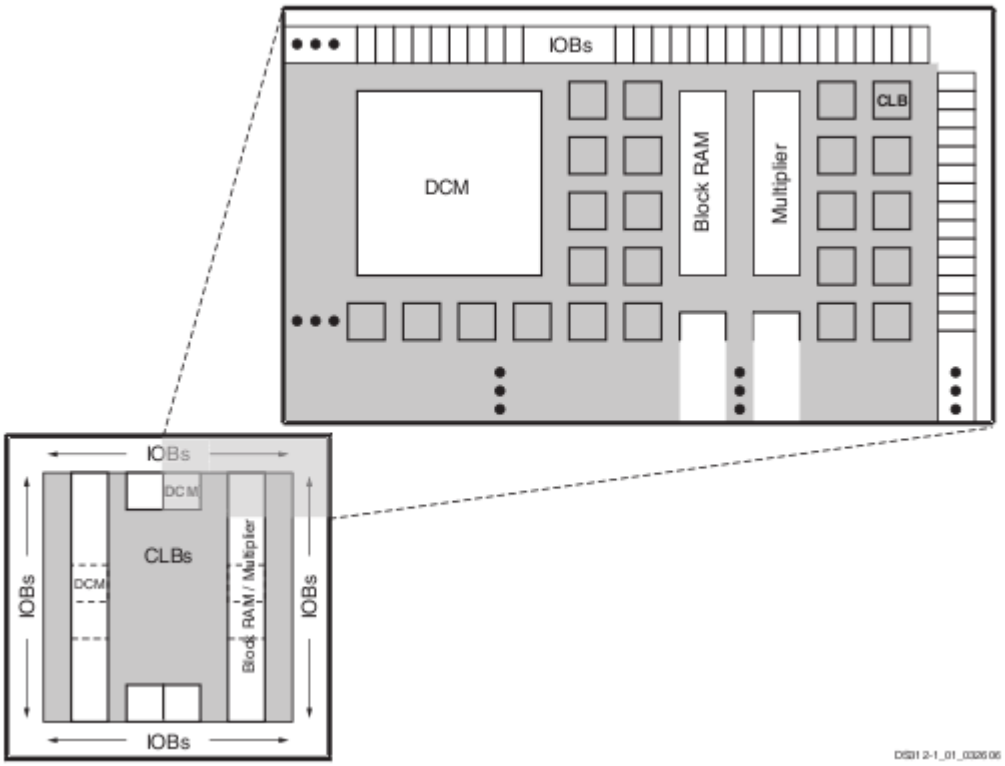


Fig.2.3 Arquitectura de un FPGA de la familia Spartan 3 (Fuente: Guía de usuario de la familia Spartan 3 [8] pág. 35)

2.2.2. Herramientas Software para la programación del FPGA

El software ISE®10.1 y EDK®10.1 son proporcionados de manera gratuita por el fabricante de FPGA Xilinx® previa inscripción en su página web.

El software ISE 10.1: es un entorno integrado de desarrollo (IDE) de circuitos y sistemas digitales usando VHDL, para este fin proporciona un conjunto de herramientas que sirven para la síntesis, depuración, implementación y configuración del FPGA.

El software EDK 10.1: es un entorno integrado de desarrollo de circuitos y sistemas digitales basado en procesadores, esta IDE proporciona un conjunto de herramientas las cuales sirven para: generar un procesador soft en el FPGA, programación del procesador soft y creación o adición de módulos IP Core propietarios o no.

a) Xilinx ISE ™

El Software Xilinx ISE 10.1 es proporcionado por el fabricante Xilinx® de manera gratuita, este sirve para diseñar e implementar circuitos y sistemas digitales en los FPGA que produce Xilinx.

b) Xilinx Platform Studio XPS

El aplicativo XPS perteneciente a EDK permite la implementación del microprocesador MicroBlaze™ en el FPGA, este microprocesador *soft* de 32 bits está basado en RISC con 32 registros de 32 bits con acceso de memoria para datos e instrucciones separados.

La memoria RAM necesaria para almacenar los datos e instrucciones es implementada dentro del FPGA usando un Bloque RAM (BRAM) que dependiendo del dispositivo puede ir de 8Kb a 64Kb.

2.3. Redes de transmisión de datos

Las redes de comunicación basan la resolución de problemas físicos y técnicos de un sistema de comunicación dividiéndolo en capas, siendo una capa o nivel una abstracción que agrupa distintos problemas para comunicar equipos entre sí. Entre estas capas o niveles podemos diferenciar: la capa física, la capa de enlace, la capa de red, la capa de transporte y la capa de aplicación.

El protocolo Ethernet, el cual es también conocido como protocolo IEEE 802.3, es el encargado de solucionar los problemas de la capa física y de enlace.

El protocolo IP, el cual está definido [9], es el encargado de solucionar los problemas de la capa de red.

En la capa de transporte tenemos los protocolos TCP y UDP, el primero es más fiable que el segundo, sin embargo, el protocolo UDP proporciona una comunicación más sencilla y puesto que la longitud de cabecera ocupa menos bits este protocolo es más rápido, y más importante aún, ya que UDP está orientado a datagramas su uso es ideal en un sistema de comunicación en tiempo real.

Finalmente se presenta el protocolo ARP el cual es necesario para la obtención de direcciones MAC a partir de una dirección IP.

A continuación una descripción más detallada de los estándares y normas que serán usadas en este trabajo.

2.3.1. Redes Ethernet

Ethernet es un estándar que también es conocido como IEEE 802.3 y que usa el método de transmisión de datos de *Acceso Múltiple por Detección de Portadora y Detección de Colisiones* CSMA/CD [10].

Las redes Ethernet diferencian cada equipo (PC, FPGA, microcontroladores, etc.) asignándole a cada uno un número de 48 bits conocido como dirección Ethernet o también como direccionamiento físico debido a que el direccionamiento se da entre dispositivos hardware[11].

Los datos Ethernet están encapsulados en tramas de longitudes no menores a 64 bytes ni mayores a 1518 bytes (excluyendo al preámbulo y delimitador de trama) tal como se muestra en la figura, los campos en la trama son transmitidos de izquierda a derecha al igual que los bits dentro de las tramas.

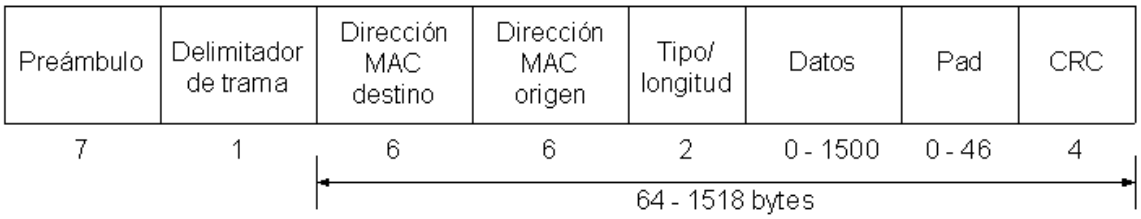


Fig.2.4 Composición de una trama Ethernet tamaño de campos en bytes (Fuente: Elaboración propia)

Preámbulo.

Este campo es usada para la sincronización y debe contener 7 bytes con el patrón "10101010", el patrón es transmitido de izquierda a derecha.

Delimitador de inicio de trama.

Este campo marca el inicio de la trama y debe contener el patrón "10101011", el patrón es transmitido de izquierda a derecha.

Dirección de destino.

Este campo es de 6 bytes de longitud y es transmitido con el LSB primero.

Dirección de origen.

Este campo es de 6 bytes de longitud y es transmitido con el LSB primero.

Tipo/Longitud.

Este campo es de 2 bytes de longitud, y sirve para especificar el protocolo usado en el siguiente nivel de la capa de abstracción, en este caso sería 0x0800 para IPv4, 0x86d0 para IPv6 y 0x0806 para ARP.

Data.

Este campo puede tener una longitud desde 0 bytes a 1500 bytes. Este campo es transmitido con el LSB primero. Para el caso de transmisión este campo debe ser siempre

proveído y para el caso de recepción este campo es siempre conservado en el paquete de datos.

Pad.

Este campo varia de 0 a 46 bytes de longitud, y es usado para asegurar que la longitud de la trama sea al menos de 64 bytes en longitud (no se considera ni el preámbulo ni el campo FCS). Para el caso de transmisión este campo es insertado automáticamente y siempre retenido en el caso de recepción.

FCS.

Este campo es de 4 bytes de longitud, el valor de este campo es calculado sobre los campos dirección destino, dirección origen, tipo, dato y pad usando CRC 32. Para el caso de transmisión este campo es insertado automáticamente y es siempre conservado en el paquete de datos recibidos.

2.3.2. Protocolo IP

Este protocolo, que fue desarrollado por el Departamento de Defensa de los EEUU [9], está limitado a proporcionar las funciones necesarias para enviar un paquete de bits (datagrama de internet) desde un origen a un destino, los cuales deben estar identificados por direcciones de longitud fija denominadas direcciones IP.

Cabe mencionar que no existen mecanismos para aumentar: la fiabilidad de datos entre los extremos, control de flujo, acuses de recibo, control de errores para datos, retransmisiones ni secuenciamiento; pero si permite proporcionar varios tipos y calidades de servicios.

El protocolo IP usa cuatro mecanismos claves para implementar su servicio de entrega y recepción de paquetes: tipo de servicio, tiempo de vida, opciones y suma de control de cabecera.

El tipo de servicio, es un conjunto de parámetros que caracterizan las elecciones del servicio presentes.

El tiempo de vida, es una indicación del límite superior en el periodo de vida de un datagrama IP, es fijado por el host que envía el datagrama y reducido en los puntos a lo largo de la ruta donde es procesado, si el tiempo de vida se reduce a cero antes de que el datagrama llegue a su destino el datagrama IP es destruido.

Las opciones, proporcionan funciones de control necesarias o útiles en algunas situaciones pero innecesarias en las comunicaciones más comunes.

La Suma de Control de Cabecera, proporciona una forma de verificar que la información en el datagrama internet ha sido transmitida correctamente. Si la suma de control de cabecera falla, el datagrama internet es descartado inmediatamente por la entidad que detecta el error.

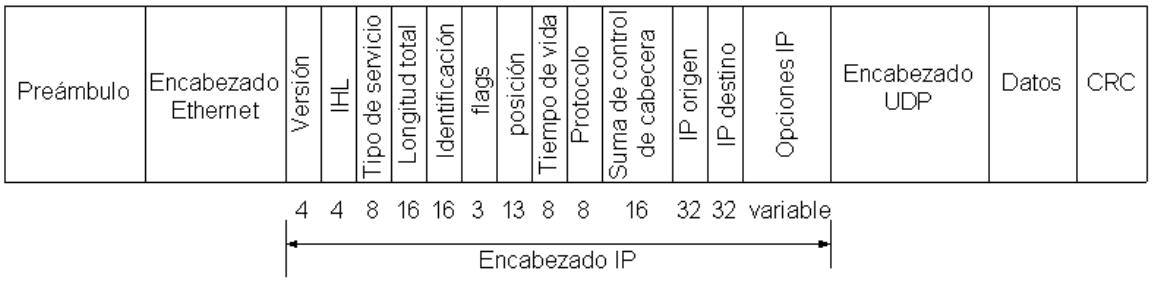


Fig.2.5 Cabecera de una trama IP (tamaño de campos en bits (Fuente: Elaboración propia)

Versión: 4 bits

El campo Versión describe el formato de la cabecera internet. Para IPV4 este campo debe tener el valor hexadecimal 0x4.

IHL: 4 bits

Longitud de la Cabecera Internet (Internet Header Length), es la longitud de la cabecera en palabras de 32 bits, y por tanto apunta al comienzo de los datos. Nótese que el valor mínimo para una cabecera correcta es 5.

Tipo de Servicio: 8 bits

Este campo proporciona una indicación de los parámetros abstractos de la calidad de servicio deseada. Estos parámetros se usarán para guiar la selección de los parámetros de servicio reales al transmitir un datagrama a través de una red en particular.

Algunas redes ofrecen prioridad de servicio, la cual trata de algún modo el tráfico de alta prioridad como más importante que el resto del tráfico (generalmente aceptando sólo tráfico por encima de cierta prioridad en momentos de sobrecarga). La elección más común es un compromiso a tres niveles entre baja demora, alta fiabilidad, y alto rendimiento.

Bits 0-2: Prioridad.

Bit 3: 0 = Demora Normal, 1 = Baja Demora.

Bit 4: 0 = Rendimiento Normal, 1 = Alto rendimiento.

Bit 5: 0 = Fiabilidad Normal, 1 = Alta fiabilidad.]

Bits 6-7: Reservado para uso futuro.

Longitud Total: 16 bits

Este parámetro es la longitud del datagrama, medida en octetos, incluyendo la cabecera y los datos.

Identificación: 16 bits

Es un valor de identificación asignado por el remitente como ayuda en el ensamblaje de fragmentos de un datagrama.

Flags (indicadores): 3 bits

Son diversos indicadores de control.

Bit 0: reservado, debe ser cero.

Bit 1: (DF) No Fragmentar

0 = puede fragmentarse,

1 = No Fragmentar.

Bit 2: (MF) Más Fragmentos

0 = Último Fragmento,

1 = Más Fragmentos.

Posición del Fragmento: 13 bits

Este campo indica a que parte del datagrama pertenece este fragmento. La posición del fragmento se mide en unidades de 8 octetos (64 bits). El primer fragmento tiene posición 0.

Tiempo de Vida: 8 bits

Este campo indica el tiempo máximo que el datagrama tiene permitido permanecer en la red. Si este campo contiene el valor cero, entonces el datagrama debe ser destruido.

Protocolo: 8 bits

Este campo indica el protocolo del siguiente nivel usado en la parte de datos del datagrama internet. En caso el siguiente nivel corresponda al protocolo UDP el número decimal 17 debería llenar este campo.

Suma de Control de Cabecera: 16 bits

Este campo almacena la suma de control de solo la cabecera. Dado que algunos campos de la cabecera cambian esta suma es recalculada y verificada en cada punto donde la cabecera internet es procesada. El algoritmo de la suma de control es:

El campo suma de control es el complemento a uno de 16 bits de la suma de los complementos a uno de todas las palabras de 16 bits de la cabecera. A la hora de calcular la suma de control, el valor inicial de este campo es cero.

Dirección de Origen: 32 bits

La dirección de origen la cual es también conocida como dirección IP origen.

Dirección de Destino: 32 bits

La dirección de destino la cual es también conocida como dirección IP destino.

2.3.3. Protocolo UDP

UDP, por sus siglas en ingles *User Datagrama Protocol*, provee un servicio de entrega de mensajes (datagramas UDP) haciendo uso de un mecanismo de procesamiento mínimo [12].

Este protocolo está orientado a transacción y no a conexión por lo que no se establece una conexión previa con el otro extremo para transmitir un mensaje UDP, esto es, no se envían ni se reciben acuses de recibo y no hay retransmisiones ni control de flujo de datos.

UDP asume que la capa de red utiliza IP, y está encapsulado en él con el identificador 17 (0x11), así también este protocolo define un conjunto de puertos denominados puertos UDP de los cuales en general se reservan del 1 al 255 para aplicaciones especificas.

Fig.2.6 Cabecera de un datagrama UDP encapsulado en un datagrama IP, tamaño de campos en bytes (Fuente: Elaboración propia)

Puerto Origen: 16 bits

Es un campo opcional y representa el puerto que usa el proceso para enviar el datagrama.

Puerto Destino: 16 bits

Número de puerto de la máquina destino, el cual es de uso obligatorio.

Longitud: 16 bits

Es la longitud en bytes del datagrama UDP incluyendo cabecera y datos, lo cual significa que el número mínimo de la longitud es 8.

Suma de verificación: 16 bits

Es la suma de comprobación de errores del mensaje, si la transmisión de este campo contiene sólo ceros entonces significa que el transmisor no ha generado esta suma.

2.3.4. Protocolo ARP

El protocolo ARP, por sus siglas *Address Resolution Protocol*, es usado cuando se quiere conocer la dirección MAC de un dispositivo teniendo como dato su dirección IP.

Antes de que IP envíe un paquete, ARP consulta una tabla local para ver si existe un mapeo directo entre la dirección IP destino de 32 bits y la dirección MAC destino de 48 bits, si no es así, ARP manda un paquete broadcast a todas las máquinas de la red en el cual solicita la dirección MAC correspondiente a la dirección IP en mención, el dispositivo con la dirección IP responde con su dirección MAC y luego IP envía el paquete.

La figura 2.7 muestra la composición de una trama ARP de solicitud o respuesta el cual incluye el encabezado Ethernet y el CRC pero sin el campo preámbulo ni inicio de trama, los números debajo de cada campo representa el tamaño en bytes de estos.

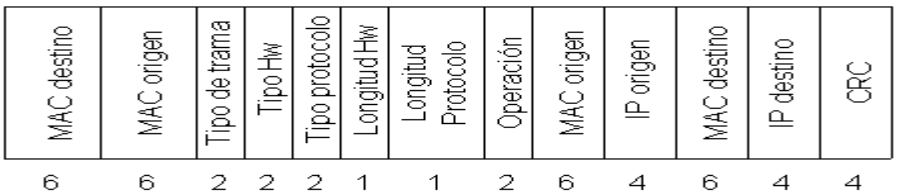


Fig.2.7 Composición de una trama ARP, tamaño de campos en bytes (Fuente: Elaboración propia)

MAC destino

Es la dirección MAC a la cual se quiere enviar la trama, en caso de no conocerse existe una dirección especial llamada broadcast para la cual todos los bits de este campo se establecen en uno.

MAC Origen

Es la dirección MAC del dispositivo desde el cual se envía la trama.

Tipo de trama

Este campo debe tener el valor hexadecimal 0x0806 cuando se trata de ARP.

Tipo HW

Es el tipo de dirección hardware usado y corresponde al valor hexadecimal 0x0001 cuando se trata de una red Ethernet.

Tipo protocolo

Es el tipo de protocolo usado en la capa de red y corresponde al valor hexadecimal 0x0800.

Longitud HW

Especifica el tamaño en bytes que tiene la dirección del hardware, en el caso de Ethernet este corresponde al valor hexadecimal 0x06

Longitud protocolo

Especifica el tamaño en bytes de la dirección del protocolo, en caso de IP este corresponde al valor hexadecimal 0x04.

Operación

Este campo especifica si la trama corresponde a una solicitud ARP (0x0001), respuesta ARP (0x0002), solicitud RARP (0x0003) o respuesta RARP (0x0004).

MAC origen

Es la dirección MAC del dispositivo que envía la trama

IP origen

Es la dirección IP del dispositivo que envía la trama

MAC destino

Es la dirección MAC del dispositivo que debe recibir la trama, en caso de no conocerse todos los bits deben ser establecidos en el valor cero.

IP destino

Es la dirección IP del dispositivo que debe recibir la trama.

2.4. Java Tiempo Real

2.4.1. Especificación para Java Tiempo Real

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystem a principios de los años 90. Este lenguaje toma mucho de la sintaxis del lenguaje de programación C y C++, sin embargo el modelo de sus objetos es más simple ya que no permite el manejo directo de memoria mediante punteros [18].

El JRE (Java Runtime Environment) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java.

Uno de los problemas más importantes de la programación en tiempo real es asegurar que la duración del tiempo de ejecución de un conjunto de secuencias de instrucción de la máquina no exceda un plazo determinado de tiempo. RTSJ introduce el concepto de *scheduling object* (planificación de objetos), esto es, es posible planificar mediante líneas de código las prioridades de las tareas a ejecutar por el CPU.

2.4.2. Implementación del programa Java Tiempo Real

En este trabajo se ha hecho uso de la implementación de TimeSys rtsj-ri-1.5, el cual requiere que los programas sean compilados por jdk 1.2, 1.3, 1.4, 1.5 ó 1.6.

La implementación de Java Tiempo Real comprende la máquina virtual Java Tiempo Real (tjvm) y un conjunto de clases que se encuentran en el paquete foundation.jar que permiten el manejo de procesos tiempo real.

a) Uso de un makefile

Aunque la línea de comandos en un terminal de un sistema operativo Linux es la forma más segura y fiable para operar la implementación RI, es evidente la posibilidad de cometer errores en el ingreso de las largas rutas de algún directorio por lo que el uso de un makefile es lo más conveniente.

b) Como compilar

Una vez terminado el makefile lo ejecutamos en un terminal (Shell), nos situamos en el directorio que lo contiene, ejecutamos la orden make y obtendremos las clases compiladas.

c) Como ejecutar

Luego de tener las clases compiladas y situados en el mismo directorio ejecutamos la orden make run con lo cual la tjvm ejecutará los procesos tiempo real.

En la figura 2.8 se muestra la secuencia de pasos para ejecutar correctamente un programa, téngase en cuenta que la limpieza es necesaria sólo cuando se ha modificado algún archivo .class del cual depende el programa.

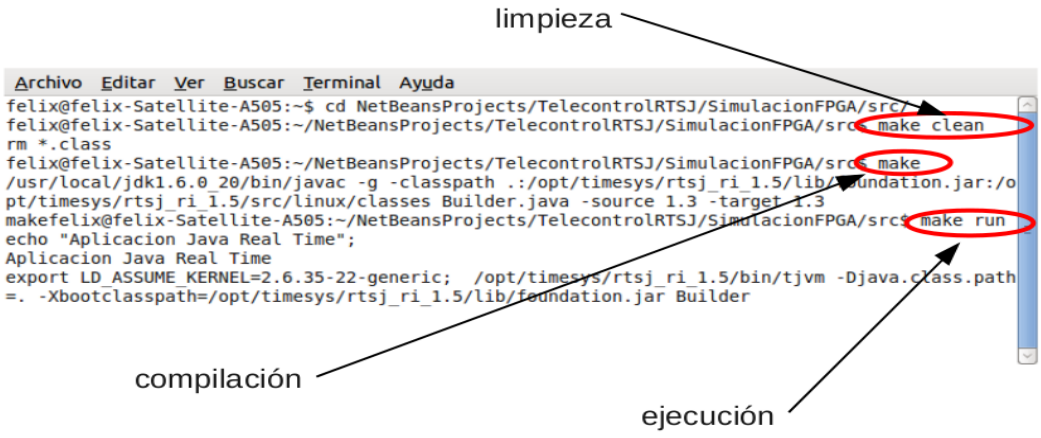


Fig.2.8 Secuencia de pasos para ejecutar un programa RTSJ

d) Como finalizar

Lo adecuado es abrir otro terminal (shell), ejecutar el comando “ps aux”, encontrar el identificador del proceso tiempo real y terminarlo con el comando “kill”.

En la figura 2.9 se muestra el proceso que debe realizarse para terminar el programa RTSJ.

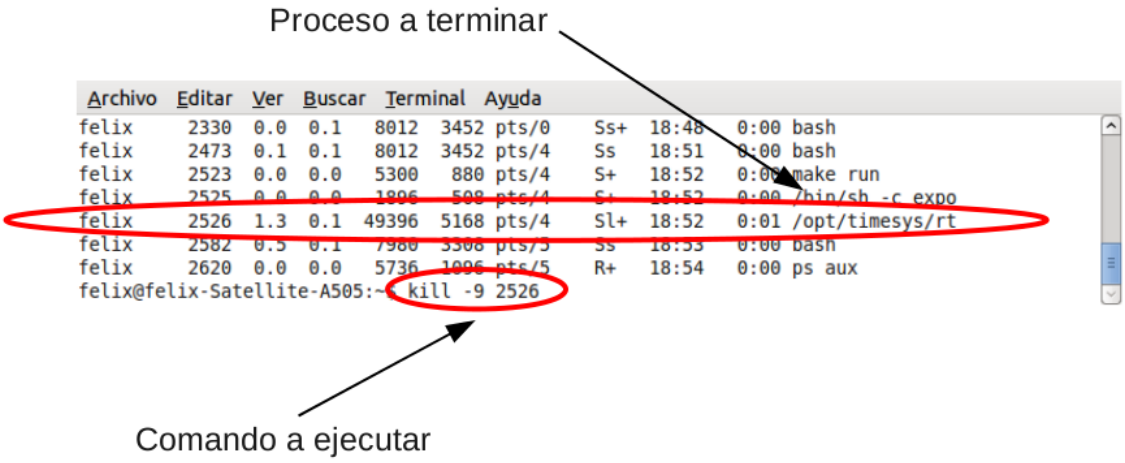


Fig.2.9 Secuencia de pasos para terminar un programa RTSJ

2.5. Convertidor DC-DC

Un convertidor DC-DC permite obtener un voltaje variable (voltaje de salida) a partir de una fuente constante de voltaje (voltaje de entrada), esto lo hace mediante el suministro y corte del voltaje de entrada el cual se logra variando los tiempos de encendido y apagado del convertidor, el tipo de convertidor capaz de realizar tales funciones es conocido como chopper.

2.5.1. Principio de operación del chopper

La modulación PWM implica entregar energía durante un cierto tiempo (tiempo de encendido) y luego cortar el suministro de ésta durante otro tiempo (tiempo de apagado) tal como se muestra en la figura 2.10.

Siendo Vs la amplitud de voltaje de la señal PWM este se aplica a la carga durante el tiempo de encendido, sin embargo para el tiempo de apagado podemos abrir los dos transistores que conducen, o cerrar sólo uno lo cual implica que podemos implementar dos esquemas diferentes que se explicarán en 2.5.3.

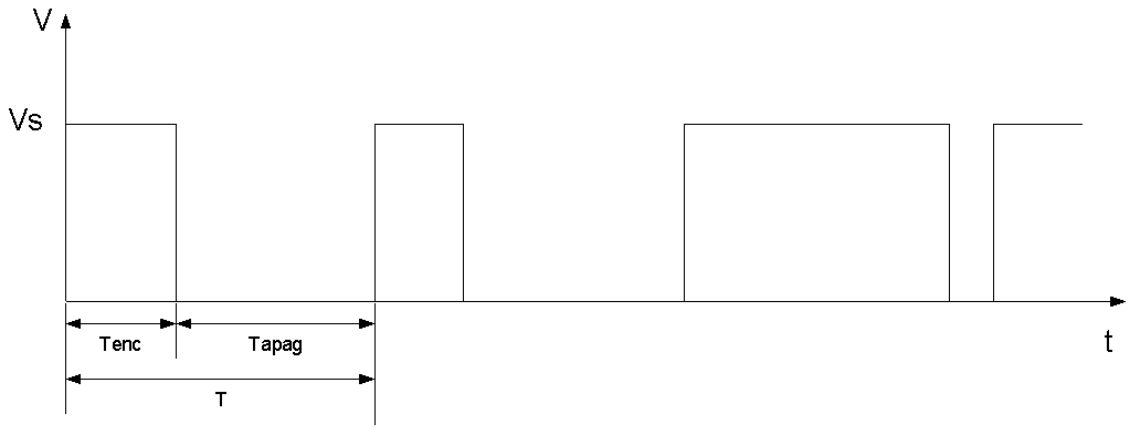


Fig.2.10 Señal PWM unipolar de frecuencia constante y ciclo de trabajo variable (Fuente: Elaboración propia)

2.5.2. Circuito chopper de 4 cuadrantes

En la figura 2.11 se muestra el diagrama esquemático de un circuito chopper implementado con dispositivos discretos BJT, los transistores Q1 y Q2 forman el semipuente izquierdo y los transistores Q3 y Q4 el semipuente derecho, este tipo de circuito es también conocido como puente H.

Cuando Q1 y Q4 conduzcan el motor girará en un sentido, si conducen Q2 y Q3 entonces el motor girará en sentido contrario, para dejar de suministrar energía al motor basta con abrir los transistores que conducen por lo que la corriente en el motor pasará por los diodos D3 y D2 o D1 y D4.

Nótese que bajo ninguna circunstancia los pares de transistores Q1 y Q4 ó Q2 y Q3 deben conducir al mismo tiempo, por lo que la electrónica que controla los transistores debe incorporar alguna manera de generar tiempos de retardos (dead time) entre el encendido y apagado de los transistores.

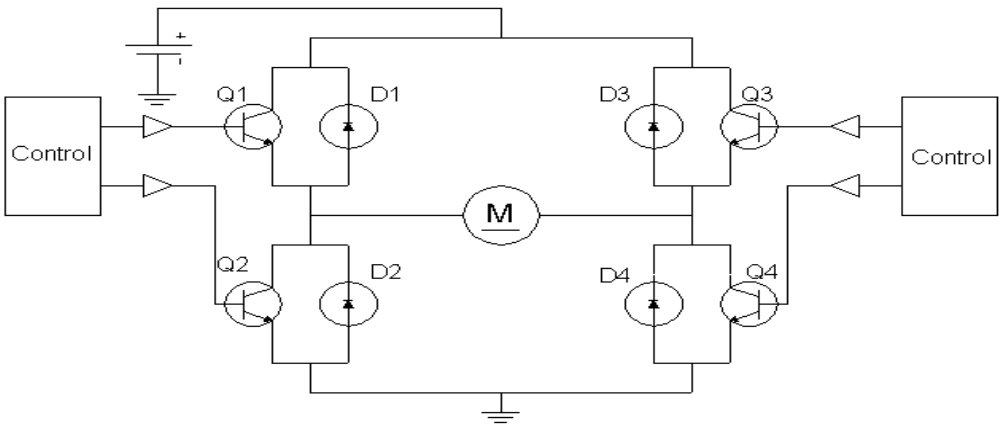


Fig.2.11 Esquema general de un puente H (Fuente: Elaboración propia)

Así también debe tenerse en cuenta la velocidad de conmutación que se requiere tengan los dispositivos de conmutación y la potencia máxima que entregarán al motor DC.

2.5.3. Tiempo de apagado

Refiriéndonos a la figura 2.12, el motor está girando debido a la corriente I que pasa por Q1 y Q4 durante el tiempo de encendido, durante el tiempo de apagado se tiene dos opciones abrir los transistores Q1 y Q4 o abrir sólo Q1 o sólo Q4.

Abrir Q1 o Q4 produce los mismos efectos, sin embargo abrir ambos produce otros efectos. Cada esquema tiene ventajas y desventajas, estas serán explicadas en lo que sigue del trabajo, sin embargo debe recordarse que para las implementaciones del capítulo 5 se hace uso del esquema diodo-transistor.

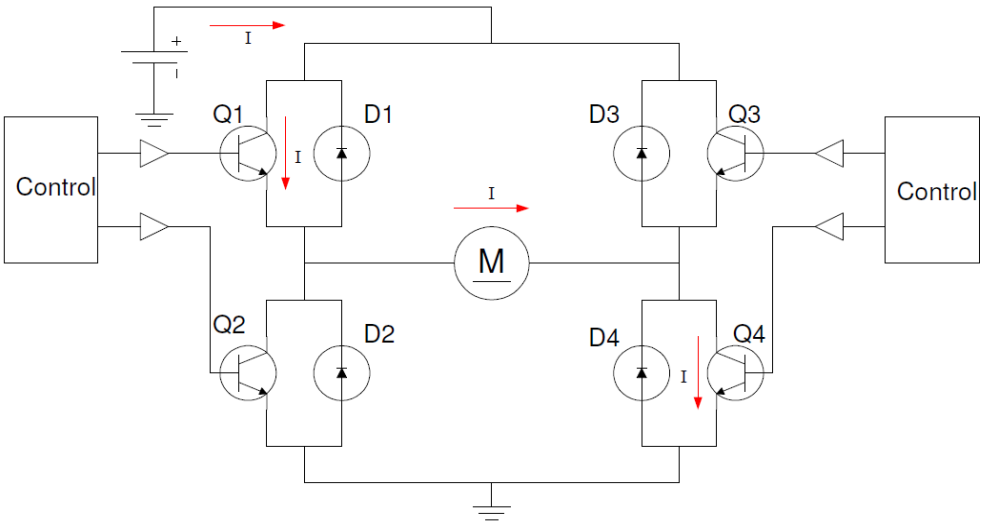


Fig.2.12 Puente H conduciendo Q1 y Q4 (Fuente: Elaboración propia)

a) Tiempo de apagado con cortocircuito, esquema diodo-transistor

Supóngase que el motor está girando debido a la corriente que pasa por los transistores Q1 y Q4 durante el tiempo de encendido tal como se muestra en la figura 2.13, y durante el tiempo de apagado abrimos sólo el transistor Q1 [19].

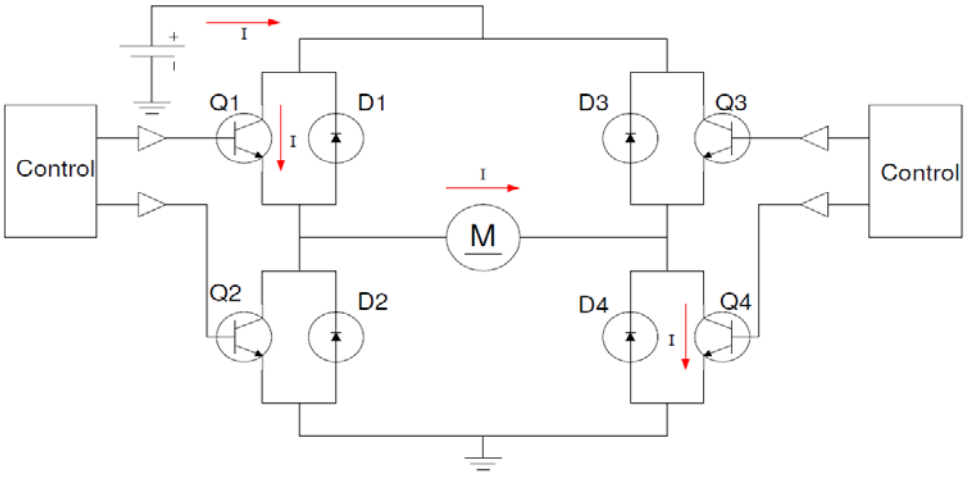


Fig.2.13 Puente H con Q1 y Q4 cerrados (Fuente: Elaboración propia)

Suponiendo que nuestro puente H cuente con D2 entonces la corriente iría por el transistor Q4 y el diodo D2 tal como se muestra en la figura 2.14.

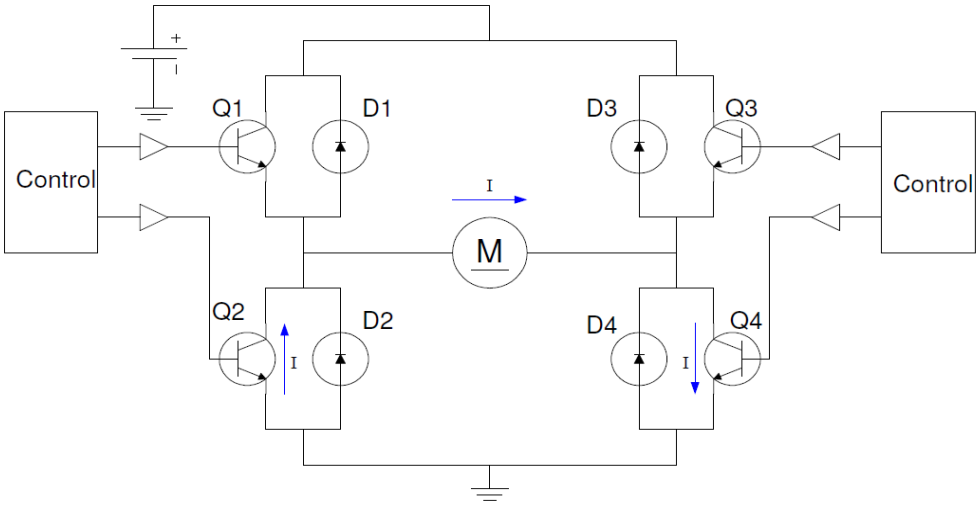


Fig.2.14 Puente H con sólo Q4 cerrado (Fuente: Elaboración propia)

Debe notarse que este esquema fuerza a que en el motor haya un voltaje que puede ser calculado como la suma del voltaje en directa del diodo y el voltaje de saturación del transistor Q4, esto hace que aún en tiempo de apagado el transistor siga consumiendo potencia y por tanto se puede afirmar que aún en tiempo de apagado el puente H sigue consumiendo potencia.

b) Tiempo de apagado con circuito abierto, esquema diodo-diodo

Supóngase el motor está girando debido a la corriente que pasa por los transistores Q1 y Q4 durante el tiempo de encendido tal como se muestra en la figura 2.15, y durante el tiempo de apagado abrimos estos transistores, de no contar con los diodos D2 y D3 la

corriente en el motor la cual no puede cambiar abruptamente produciría una elevación del voltaje hasta que este pueda formar un arco de corriente o dañe algún dispositivo [19].

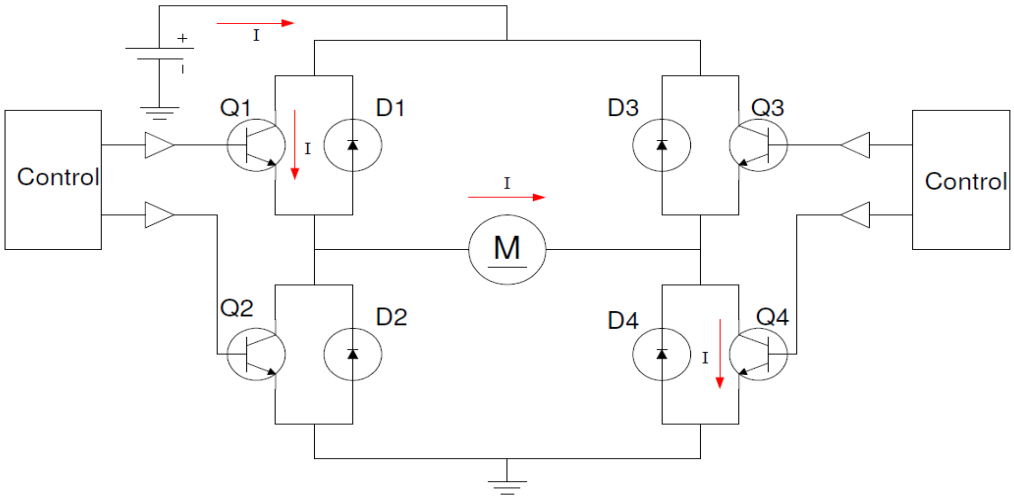


Fig.2.15 Puente H con Q1 y Q4 cerrados (Fuente: Elaboración propia)

Suponiendo que nuestro puente H cuente con D2 y D3, la corriente encontraría este camino e iría por él, tal como se muestra en la figura 2.16.

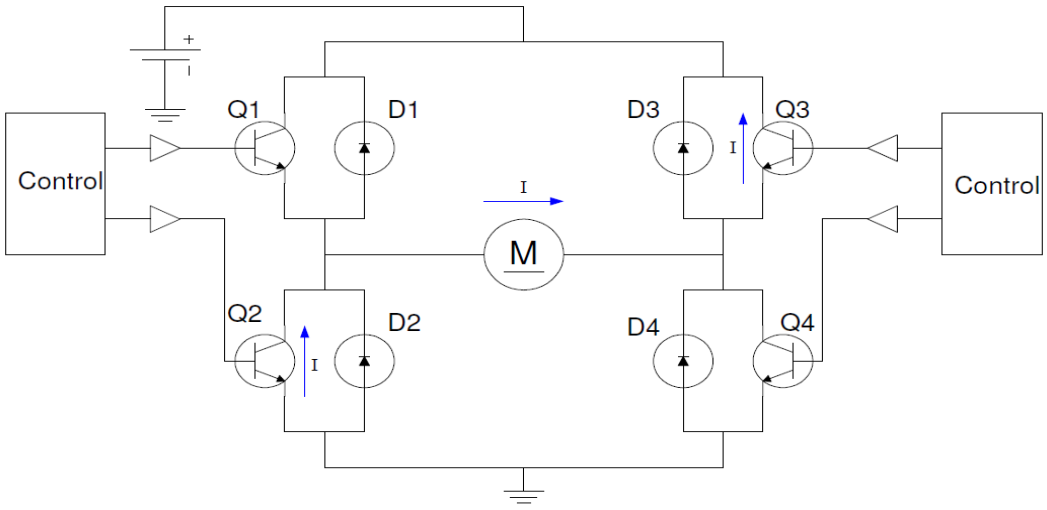


Fig.2.16 Puente H con Q1 y Q4 abiertos (Fuente: Elaboración propia)

Nótese que en este caso no se fuerza al motor a tener algún voltaje determinado, es decir, el voltaje que esté presente estará dado por la energía almacenada en la inductancia y la velocidad a la cual gire el motor, así también dado que ningún transistor consume corriente se puede afirmar que en el tiempo de apagado el puente H, en este esquema, no consume potencia.

CAPÍTULO III
DISEÑO DEL SISTEMA

3.1. El circuito integrado LAN83C185

El SMSC LAN83C185 es un circuito integrado analógico altamente integrado, de bajo consumo de potencia y de alto rendimiento para aplicaciones Ethernet embebidas. Este chip requiere sólo una única fuente de +3.3 V y es totalmente compatible con el estándar IEEE 802.3/802.3u tal como puede verse en su hoja de datos [20].

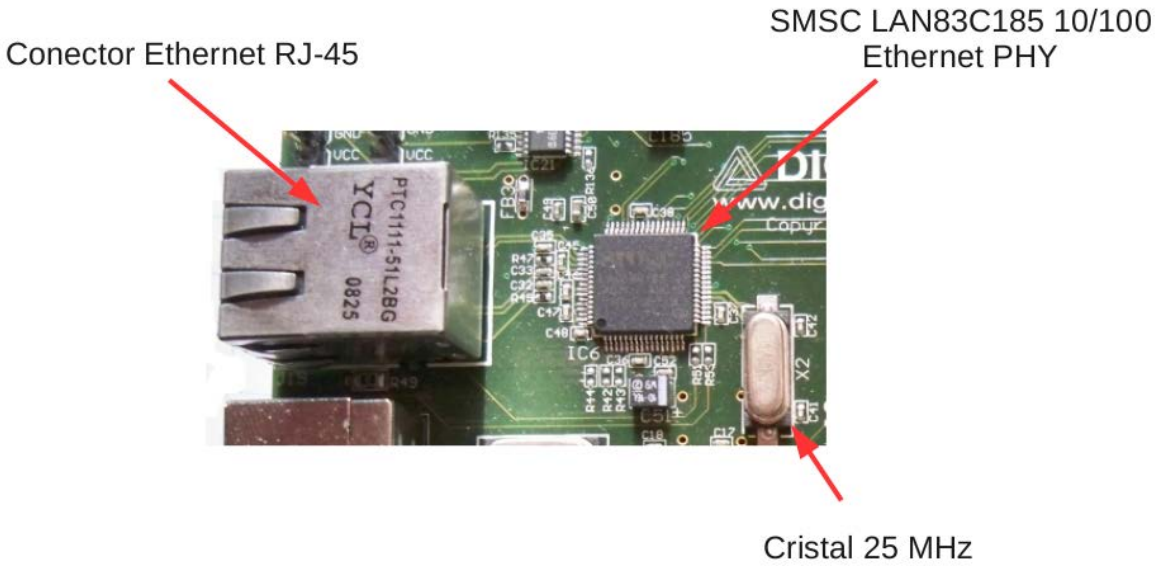


Fig.3.1 Ubicación de LAN83C185 en la tarjeta del FPGA (Fuente: Elaboración propia)

3.1.1. Características del CI LAN83C185

El LAN83C185 consiste de un codificador/decodificador, scrambler/descrambler[2], transmisor con formación de onda y driver de salida, receptor con ecualizador adaptativo y corrección baselinewander (BLW), un recuperador de datos y reloj y Media Independent Interface MII [20].

El LAN83C185 es totalmente compatible con las normas IEEE 802.3/802. Este contiene un transceptor full dúplex 10-BASET/100BASE-TX y soporta operación en 10-

[2] Scrambling, eliminación de picos grandes y de banda angosta en la densidad espectral de potencia [20]

Mbps (10BASE-T) sobre cables de par trenzado sin blindaje categoría 3 y categoría 5, así como operación en 100-Mbps (100BASE-TX) sobre cables de par trenzado sin blindaje categoría 5.

Media Independent Interface (MII) es una interface estándar de 4 bits entre el Control de acceso al medio (MAC por sus siglas en ingles) y el bloque que implementa la capa física de Ethernet (PHY), esta interface sirve como soporte para la transmisión y recepción de tramas de datos. En modo 10 Mb/s el MII debe correr a 2.5MHz; en modo 100Mb/s la MII debe correr a 25MHz.

La MII está compuesta por tres vías: una para transmisión de datos, otra vía para recepción de datos y una última para la gestión de los datos, los cuales son usados para leer y escribir en los registros PHY.

3.2. FPGA Spartan 3E

3.2.1. Características del FPGA Spartan3E XC3S500E

Este chip de silicio soporta la implementación de procesadores MicroBlaze y controladores PicoBlaze. La denominación de 500 se refiere al número compuertas lógicas es decir 500k lo que equivale a 1164 CLB (Bloques lógicos configurables) [8].

Dependiendo de cómo se configuren estos CLB's y de la cantidad disponible se puede implementar diversidad de circuitos y sistemas digitales tales como procesadores, controladores, codificadores, filtros digitales entre otros, de manera que uno de los principales parámetros a tener en cuenta cuando se diseña algún circuito o sistema en VHDL es la cantidad de CLB's que se requiere para su implementación.

El FPGA XC3S500E cuya completa descripción puede encontrarse en [8], dispone de 73Kbits de memoria RAM distribuida (Distributed RAM bits) y 20 bloques de memoria RAM (BRAM) cada uno con una capacidad de 18Kbits, siendo 1Kbit igual a 1024 bits por convenciones de memoria.

En la figura 3.2 se muestra el FPGA Spartan 3E dentro de la plataforma de desarrollo: Kit Spartan 3E elaborado por Digilent.

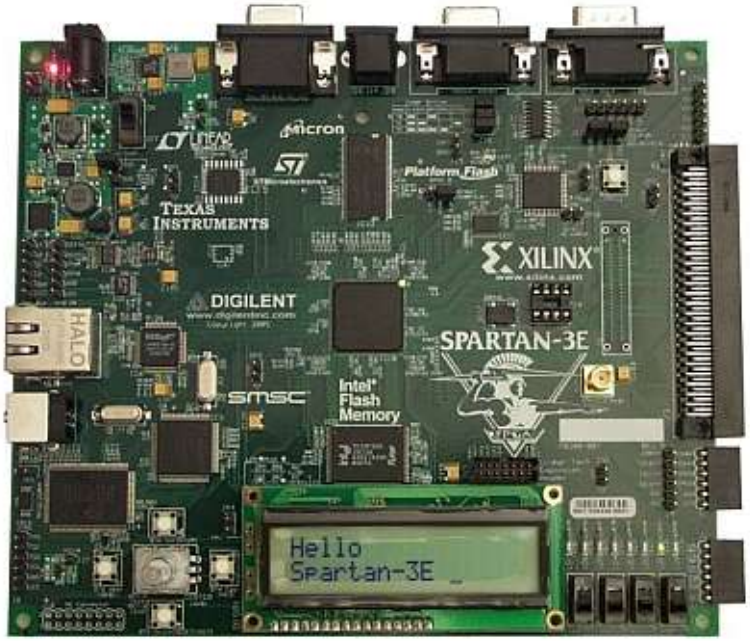


Fig.3.2 Módulo de desarrollo basado en el FPGA Spartan 3E (Fuente: Guía de usuario del kit de inicio Spartan 3E [21])

3.2.2. Configuración del FPGA

La implementación de los circuitos y sistemas en el FPGA es un proceso conocido como “download bitstream”, el termino bitstream hace referencia un conjunto de bits almacenados en el FPGA con la función de determinar cómo deben configurarse los CLB’s.

El proceso de descarga del bitstream (en adelante .bit) puede hacerse usando tanto EDK 10.1 como ISE 10.1. Usando EDK 10.1 el proceso se realiza haciendo la conexión USB-JTAG y luego pulsando el botón download bitstream ubicado en la barra de herramientas del IDE tal como se muestra en la figura 3.1.

Fig.3.3 Botones de barra de herramientas Hardware y Software (Fuente: Elaboración propia)

El principal destino de un .bit es su descarga al FPGA, sin embargo, es a veces necesario que la información del .bit quede almacenada en algún otro dispositivo con memoria no volátil, de manera que el FPGA pueda siempre tener un respaldo de donde obtener información para su configuración sin necesidad de hacerlo manualmente.

Para este propósito la tarjeta del FPGA dispone de una memoria flash SPI y una memoria flash PROM que pueden servir como almacenamiento del archivo .bit.

Los pasos a seguir para la descarga del .bit a una de estas memorias se encuentra explicado en [22].

3.2.3. Programación del microprocesador MicroBlaze

Los principales componentes y módulos que estarán conectados al procesador son:

- El bus local del procesador: PLBv4.6 que es el bus local del microprocesador (mb_plb)
- El bus local de memoria: LMBv1.0 del cual se instancian dos de estos uno para datos dlmb y otro para instrucciones ilmb,
- El bloque de RAM: BRAMv1.0
- El componente XPS 10/100 Ethernet MAC Lite:
- El componente XPS INTC
- El componente XPS TIMER
- La IP core PWM
- La IP core ADQ_ENCODER

La conexión de todos estos componentes puede verse en la figura 3.4

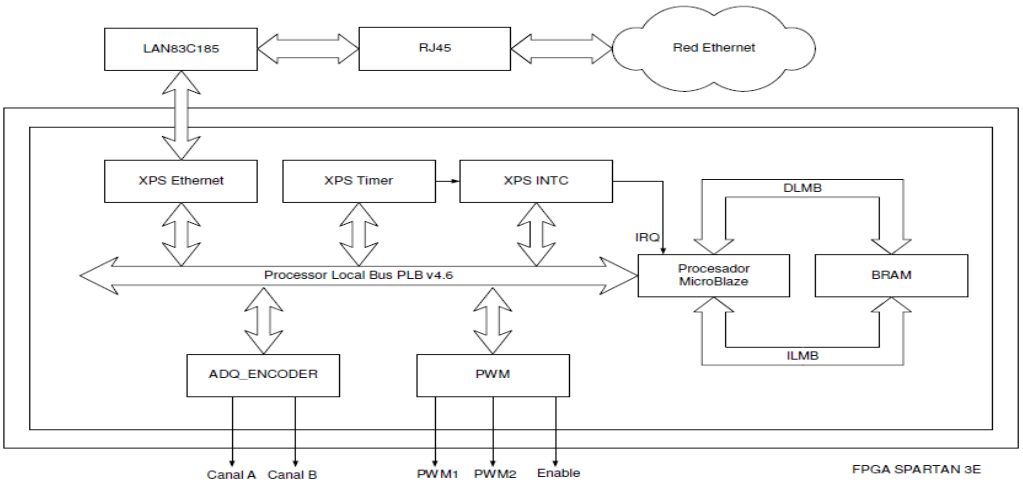


Fig.3.4 Visión Global de los periféricos que se conectarán al procesador MicroBlaze (Fuente: Elaboración propia)

a) El procesador MicroBlaze® v7.1

El procesador embebido MicroBlaze® es un computador con un conjunto de instrucciones reducidas (RISC), el cual está optimizado para la implementación en FPGA's de Xilinx®.

Este procesador soporta una fuente de interrupción externa (conectado al puerto de entrada *Interrupt*), al cual reacciona sólo si el bit *Enable Interrupt* (IE) del registro de estados de la máquina (MSR) está establecido en 1.

Al ocurrir una interrupción, la instrucción en el estado de ejecución se completa, mientras que la instrucción en el estado de decodificación es reemplazado por el vector de interrupción (dirección 0x10). La dirección de retorno de la interrupción es automáticamente cargada al registro de propósito general R14.

Las interrupciones son ignoradas por el procesador si cualquiera de los bits, BIP (Break in progress) o EIP (Exception in progress), en el MSR están establecidos a 1.

b) El componente XPS 10/100 Ethernet MAC Lite v2.0

El Ethernet Lite MAC (Media Access Controller) soporta la Media Interface Independent (MII) para la comunicación con un dispositivo de la capa física (Physical Layer) descrita en el estándar IEEE 802.3, así también se comunica con el procesador usando la interface ofrecida por el Processor Local Bus (PLB). La documentación respecto de esta IP core se puede encontrar en [10].

El diseño provee una interface de 10Mbps y 100Mbps, una interface PLB basado en la especificación PLB v4.6, MII para conexión con transceptores externos de la capa física, Un puerto dual de memoria de 2K byte destinados a retener los datos de un paquete para transmisión y recepción.

Interface de transmisión.

Los datos a transmitir deben ser almacenados en el puerto dual de memoria iniciando los primeros 4 bytes en la dirección C_BASEADDRESS +0x0000, los segundos 4 bytes en la dirección C_BASEADDRESS +0x0004 y así sucesivamente con los siguientes datos, asi también debe mencionarse que la interface de 32 bits requiere que los 4 bytes sean escritos de una sola vez puesto que no existe habilitación de bytes individuales en una palabra de 32 bits.

Los datos de transmisión deben incluir los campos de dirección destino (6 bytes), dirección origen (6 bytes), tipo/longitud (2 bytes) y datos (0 - 1500 bytes) además de que estos deben ser almacenados en la memoria de manera contigua. El preámbulo, inicio de trama y CRC no deben ser incluidos en el puerto dual de memoria, ya que estos son añadidos automáticamente.

La dirección del puerto dual de memoria C_BASEADDRESS +0x07F8 es usado para establecer el bit de Habilitación Global de Interrupciones (GIE). La generación de pedidos de interrupción se habilita estableciendo GIE=’1’ y se deshabilita estableciendo GIE=’0’.

La dirección del puerto dual de memoria C_BASEADDRESS +0x07F4 es usado para almacenar la longitud en bytes de los datos almacenados en el puerto dual de memoria, los 8 bits más significativos del valor de la longitud deben ser almacenados en los bits de 16-23 mientras que los 8 bits menos significativos deben ser almacenados en los bits 24-31.

Tres bits de la dirección del puerto dual de memoria C_BASEADDRESS +0x07FC son usados, el bit 31 representa el bit Status, el bit 30 representa el bit Program y el cuarto bit es decir el bit 28 el bit de habilitación de interrupción.

La IP Ethernet Lite MAC requiere que la longitud de los datos a transmitir sean almacenados en la dirección C_BASEADDRESS +0x07F4 antes de que se establezca el bit Status de la dirección C_BASEADDRESS +0x07FC, la secuencia para iniciar una transmisión es como sigue.

El software almacena los datos a transmitir en el puerto dual de memoria iniciando en la dirección C_BASEADDRESS +0x0

El software escribe la longitud en bytes de los datos en la dirección C_BASEADDRESS +0x07F4.

El software escribe un ‘1’ en el bit Status en la dirección C_BASEADDRESS +0x07FC (bit 31 del bus de datos).

El software espera que el bit Status se establezca en ‘0’ antes iniciar otra transmisión.

Si el bit 0 de la dirección C_BASEADDRESS +0x07F8 (GIE) y el bit 28 de la dirección C_BASEADDRESS +0x07FC están establecidos a ‘1’ entonces ocurrirá una interrupción cuando la IP establezca en ‘0’ el bit Status.

Cuando se establece el bit Status a ‘1’ se inicia la transmisión la cual realiza las siguientes funciones:

- Genera los campos de preámbulo e inicio de trama.

- Lee la longitud y especifica la cantidad de datos que deben leerse del puerto dual de memoria así como añadir padding si es requerido.
- Detecta alguna colisión
- Calcula el CRC y
- Limpia el bit Status cuando se completa la transmisión

Interface de recepción

De toda la trama recibida se almacena desde la dirección destino hasta el CRC en el área de memoria del puerto dual de recepción el cual inicia en la dirección C_BASEADDRESS +0x1000.

De la dirección de memoria 0x17FC es usado el bit 31 para representar la presencia de una trama recibida que se encuentra lista para ser procesada por el software, este bit es denominado bit de estado (Status bit).

De la dirección de memoria 0x17FC es usado el bit 28 para la habilitación de interrupciones, si este bit y el bit GIE se encuentran activados entonces se generara un pulso cuando la memoria tenga datos disponibles sin importar el momento.

Cuando el bit de estado es ‘0’, esta IP core esperara por un paquete Ethernet cuya dirección destino coincida con la dirección MAC del FPGA o tenga dirección broadcast, en ambos casos el paquete es almacenado en el puerto dual de memoria iniciando en la dirección 0x1000.

Una vez recibido el paquete esta IP core verifica el CRC, si el valor del CRC es correcto el bit de estado se establece en ‘1’, en caso contrario se descarta el paquete y se sigue a la espera de otro.

Si el bit de estado se encuentra en ‘1’ la IP core no realizara ninguna otra operación de recepción hasta que el bit de estado sea establecido por el software en ‘0’ indicando que los datos han sido recuperados del puerto dual de memoria.

En resumen se puede decir que:

- El software monitorea el bit de estado hasta que este sea establecido en ‘1’ por la IP core.
- Una vez el bit de estado esté en ‘1’ u ocurra una interrupción que indique una recepción completa el software lee la trama entera y la almacena en alguna otra dirección de memoria especificada en tiempo de compilación.

- El software escribe ‘0’ en el bit de estado, de manera que la IP core quede habilitada para recibir algún otro paquete.

c) El componente XPS Interrupt Controller v1.00a

Este componente proporcionado por Xilinx concentra múltiples entradas de interrupción de dispositivos periféricos y genera una salida de interrupción al procesador.

El diseño del sistema basado en la atención de interrupciones es necesario tanto para el proceso de modelamiento como de control, en ambos procesos se requiere un intervalo de muestreo fijo que puede ser obtenido por un timer cuya salida esté conectada a este componente, de otro modo tendría que implementarse la periodicidad mediante un bucle *while* o *for* que mantendrían ocupado al procesador en vez de que este pueda aprovechar el tiempo en realizar otras tareas.

La documentación respecto de esta IP se puede encontrar en [13]

d) El componente XPS Timer / Counter v1.00a

Este componente proporcionado por Xilinx es un módulo timer de 32 bits conectado al bus PLB, este módulo está compuesto por dos timer cuyos intervalos de tiempo tienen las capacidades de generar interrupción, capturar eventos y generar PWM.

La documentación respecto de esta IP core se puede encontrar en [14]

3.2.4. Documentación Application Program Interface (API)

El software EDK v10.1 provee la documentación necesaria para manejar algunas IP que ofrece de manera gratuita.

a) Ethernet Lite MAC

De toda la documentación que ofrece el fabricante Xilinx, es el archivo de cabecera xemaclite.h, el cual puede encontrarse en [15], el más importante ya que implementa algunas funciones y estructuras que elevan el nivel de abstracción al momento de hacer el software que manejará la IP en mención.

En el archivo de referencia de xemaclite.h [15] tenemos las siguientes funciones:

- Int XEmacLite_Initialize (XEmacLite* InstancePtr, u16 DeviceID)
- Void XEmacLite_SetMacAddress(XEmacLite* InstancePtr,u8* AddressPtr)
- Void XEmacLite_FlushReceive(XEmacLite* InstancePtr);

- Int XEmacLite_Send(XEmacLite * InstancePtr,u8 * FramePtr,unsigned ByteCount)
- U16 XEmacLite_Recv(XEmacLite* InstancePtr,u8* FramePtr)

Así también xemaclite.h provee una estructura llamada XEmacLite la cual es una instancia que posee todos los componentes necesarios para manejar la IP.

Int XEmacLite_Initialize (XEmacLite* InstancePtr, u16 DeviceID)

Esta función inicializa los campos de una estructura XEmacLite que debe hacer sido creada antes de la llamada a esta función.

Parámetros:

InstancePtr: Es un puntero a la instancia XEmacLite.

DeviceId: Es la única id del dispositivo controlado por esta instancia

Retorna:

_ XST_SUCCESS: Si la inicialización se realizó exitosamente.

_ XST_DEVICE_NOT_FOUND/XST_FAILURE: Si la información de la configuración del dispositivo no fue encontrada para un dispositivo con la id suministrada.

Void XEmacLite_SetMacAddress (XEmacLite* InstancePtr,u8* AddressPtr)

Establece la dirección MAC de 48 bits del dispositivo,

Parámetros:

InstancePtr: Es un puntero a la instancia XEmacLite.

AddressPtr: Es un puntero a la dirección MAC de 6 bytes, el orden es del mayor byte al menor.

Retorna:

Nada

Void XEmacLite_FlushReceive (XEmacLite* InstancePtr)

Limpia el buffer de recepción

Parámetros:

InstancePtr: Es un puntero a la instancia XEmacLite de la cual se borraran los datos del buffer de recepción.

Retorna:

Nada

Int XEmacLite_Send(XEmacLite * InstancePtr,
u8 * FramePtr,
unsigned ByteCount)

Envía una trama Ethernet, ByteCount es el tamaño total de la trama incluyendo la cabecera.

Parámetros:

InstancePtr: Es un puntero a la instancia XEmacLite.

FramePtr: Es un puntero a la trama. Para un rendimiento óptimo, un buffer alineado de 32 bits debe ser usado, aunque esto no es requerido ya que la función alineará los datos de ser necesario.

ByteCount: Es el tamaño en bytes de la trama.

Retorna:

_ XST_SUCCESS: Si la inicialización se realizó exitosamente.

_ XST_DEVICE_NOT_FOUND/XST_FAILURE: Si el(los) buffer(s) están llenos o no tiene datos validos.

U16 XEmacLite_Recv (XEmacLite* InstancePtr, u8* FramePtr)

Recibe una trama

Parámetros:

InstancePtr: Es un puntero a la instancia XEmacLite.

FramePtr: Es un puntero a un buffer donde la trama será almacenada, el buffer debe ser al menos XEL_MAX_FRAME_SIZE bytes. Para un rendimiento óptimo un buffer alineado de 32 bits debe ser usado, aunque esto no es requerido ya que la función alineará los datos de ser necesario.

Retorna:

El campo type/length de la trama recibida, o retorna ‘0’ si no hay datos esperando en el buffer de recepción.

b) XPS Interrupt Controller

Al igual que el componente anterior las principales funciones se encuentran en el archivo de cabecera xintc.h, sin embargo se usará en la configuración de este componente el archivo xintc_l.h, este ofrece funciones y macros a bajo nivel lo que proporciona un acceso y control total a este dispositivo, la documentación respectiva puede encontrarse en [16].

De todas las funciones implementadas son tres las necesarias para nuestros propósitos:

Void XIntc_RegisterHandler (u32 BaseAddress,
int InterruptID,
XinterruptHandler Handler,
void * CallBackRef)

Esta función registra a otra función que será llamada cuando se atienda una interrupción solicitada por algún dispositivo.

Parámetros:

BaseAddress: Es la dirección de memoria del controlador de interrupción.

InterruptId: Es la identificación asociada con algún dispositivo que solicita una interrupción.

CallBackRef: Es el argumento que será pasado a la función que manejará la interrupción.

Retorna:

Nada

Void XIntc_mMasterEnable (u32 BaseAddress)

Esta función habilita todas las interrupciones en el registro de habilitación maestro (Master Enable Register).

Parámetros:

BaseAddress: Es la dirección de memoria del controlador de interrupción.

Retorna:

Nada

Void XIntc_mEnableIntr (u32 BaseAddress, u32 EnableMask)

Esta función habilita la entrada de interrupción de algún dispositivo en especifico.

Parámetros:

BaseAddress: Es la dirección de memoria del controlador de interrupción.

EnableMask: Es un valor de 32 bits que se escribirá en el registro de habilitación, cada bit de la máscara corresponde a una señal de entrada que solicitará una interrupción (INT0 = LSB).

Retorna: Nada

c) XPS Timer/Counter

Al igual que el componente anterior las principales funciones se encuentran en el archivo de cabecera xintc.h, sin embargo para la configuración de este componente se usará el archivo xintc_l.h, el cual ofrece funciones y macros a bajo nivel lo que proporciona un acceso y control total a este dispositivo, la documentación respectiva puede encontrarse en [17].

De todas las funciones implementadas son tres las necesarias para nuestros propósitos.

Void XIntc_RegisterHandler (u32 BaseAddress,
int InterruptID,
XinterruptHandler Handler,
void * CallBackRef)

Esta función registra una función que será llamada cuando se atienda una interrupción solicitada por algún dispositivo.

Parámetros:

BaseAddress: Es la dirección de memoria del controlador de interrupción.

InterruptId: Es la identificación asociada con algún dispositivo que solicita una interrupción.

CallBackRef: Es el argumento que será pasado a la función que manejará la interrupción.

Retorna:

Nada

3.2.5. Creación de IP usando EDK

La principal ventaja de un FPGA es que su arquitectura interna no está totalmente fija ya que las conexiones entre CLB se pueden modificar y por tanto se puede crear los circuitos que se deseen.

Por esta razón el fabricante Xilinx provee no sólo su procesador MicroBlaze, si no que brinda un conjunto de IP cores que sirven de periféricos, interface o coprocesamiento al MicroBlaze.

Sin embargo puede darse el caso que las IP cores de Xilinx no satisfacen las necesidades de los diseñadores o que estos últimos deseen tener pleno control de los circuitos a implementar y entonces sólo queda hacer uso de las plantillas brindadas por Xilinx y desarrollar nuestras propias IP cores [23].

En este trabajo se han creado dos IP core las cuales no ejecutan pedidos de interrupción, no tiene reset por software y son conectadas en configuración esclavo al Processor Local Bus v4.6 tal como se muestra en Figura 3.5.

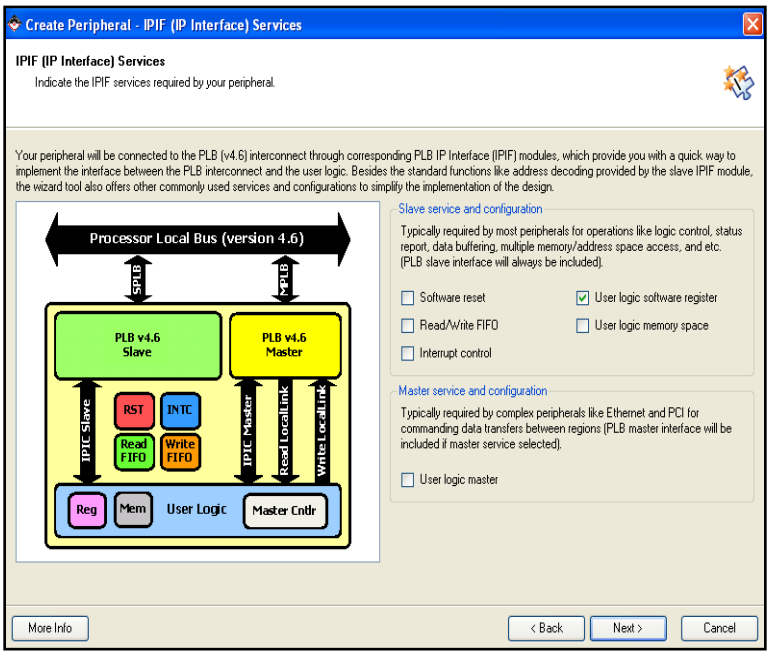


Fig.3.5 Selección de servicios e interfaces para la IP (Fuente: Software EDK 10.1)

La figura 3.6 muestra un diagrama de bloques de lo que el software EDK crea, esto es, nos da la interface para comunicar nuestra IP core con el PLB, y nuestro bloque VHDL se implementa en el rectángulo de azul denominado lógica de usuario (user logic).

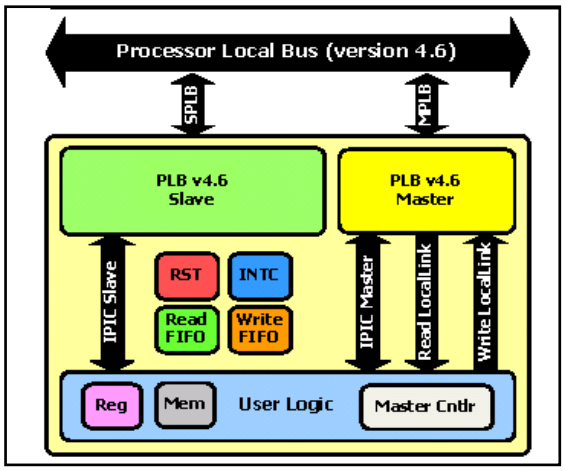


Fig.3.6 Diagrama de conexionado IP (Fuente: Software EDK 10.1)

3.3. Periféricos externos

3.3.1. Circuito de potencia para el control de un motor DC

El control de la velocidad de un motor puede hacerse de manera unidireccional y bidireccional, cuando el control es unidireccional el circuito de potencia es bastante sencillo y se resume al uso de un conmutador de potencia (MOSFET o BJT) mientras que cuando se desea un control bidireccional el circuito de potencia por lo menos debe contar con cuatro conmutadores de potencia.

Según las necesidades en la velocidad de conmutación y disipación de potencia disponemos de la tecnología BJT, MOSFET e IGBT; para bajas potencia y baja velocidad de conmutación basta con el uso de BJT, para mediana potencia y alta velocidad de conmutación se recomienda el uso de MOSFET y para altas potencias y mediana velocidad de conmutación se recomienda el uso de IGBT.

La tabla 3.1 muestra las características del motor de manera que nuestra aplicación es de baja potencia y es posible hacer uso de tecnología BJT específicamente hacemos uso del IC L298N que en sus interior dispone de los cuatro conmutadores (transistores BJT) que conforman el puente H, este IC puede alimentar cargas de hasta 46 Volt y 2 A con una velocidad de conmutación máxima de 25 kHz [24].

Motor DC Hitachi	
Voltios 30	Output 42W
Amperios 2.0	Encoder 400P/R
RPM 2750	Tipo D06D401E

TABLA Nº 3.1. Características de motor DC Hitachi

Para el funcionamiento del CI L298 se requieren dos fuentes de alimentación, una para su lógica interna y otra para la alimentación de la carga (motor DC).

Es de uso vital 4 diodos de bajo trr (time recover reverse) que aseguren un camino para la corriente cuando los conmutadores del puente H bloqueen el paso de esta.

Dado que el FPGA provee las señales de control al CI L298 se decidió hacer uso de optoacopladores que aíslen la etapa de control de la etapa de potencia y así proteger al FPGA de cortocircuitos o errores en la manipulación del modulo.

Los optoacopladores elegidos tienen una velocidad de conmutación de hasta 1MHz y con salida digital TTL (salida de un Schmitt Trigger) lo que nos permite se pueda conectar

directamente al IC L298 (entrada digital TTL) sin necesidad de acondicionar la señal, estos optoacopladores son de la familia H11L1M [25].

3.3.2. Medición de Velocidad y Sentido de giro.

El funcionamiento de un encoder se basa en la interrupción de un haz de luz continuo, por lo que contando el número de pulsos en una ventana de tiempo se puede determinar la velocidad del eje del motor.

Un encoder se encuentra constituido por un disco opaco con agujeros dispuestos radial y angularmente equidistantes, una fuente luminosa y un sistema fotoreceptor tal como se muestra en la figura 3.7.

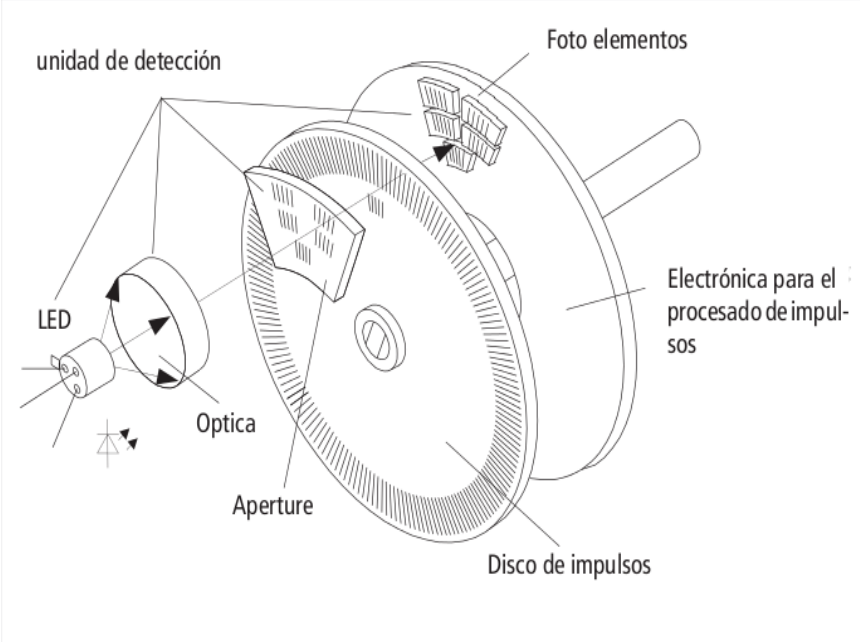


Fig.3.7 Funcionamiento de un encoder (Fuente: www.tr-electronic.de)

El eje cuya posición se quiere medir va acoplado al disco, de manera que a medida que el eje gire se interrumpa el haz de luz, es fácil deducir que la resolución depende del número de agujeros que posea el disco.

Dado que la salida del optoacoplador del encoder es una señal analógica (señal de un fotodiodo) esta debe ser tratada antes de ser procesada por el FPGA, es decir el tren de pulsos analógico del encoder debe convertirse en un tren de pulsos digital.

Para este tratamiento se usará un disparador Schmitt que mediante su ciclo de histéresis convertirá nuestra señal analógica a una digital.

Es bastante conocido el esquema de adquisición en el que se cuentan los pulsos en una ventana de tiempo y que mediante una regla de tres simples se obtiene la velocidad.

En este trabajo se decidió probar un nuevo esquema en el que en vez de contar los pulsos del encoder se va a medir el intervalo de tiempo entre cada flanco de subida.

a) Esquema de intervalo de tiempo entre flancos de subida

Este esquema se basa en medir el tiempo entre dos flancos de subida de manera que la velocidad queda determinada por:

$$\omega = \frac{\Delta\theta}{\Delta t} \tag{3.1}$$

Dónde:

$\Delta\theta$: representa el ángulo en radianes entre dos agujeros consecutivos, si por ejemplo la resolución del encoder es de 400 agujeros entonces $\Delta\theta = 2\pi/400$.

Δt: representa el intervalo de tiempo medido entre cada flanco de subida del tren de pulsos.

Este esquema se hacía prometedor ya que aumentaba la resolución de la medición tal como se puede observar en las siguientes líneas.

Reemplazando la resolución del encoder:

$$\omega = \frac{1rev}{400 \cdot \Delta t(s)} \frac{60s}{1min} \tag{3.2}$$

De donde es fácil observar que la resolución de la velocidad angular que se pueda medir queda limitada a la resolución de la medición del tiempo el cual es medido contando cuantos pulsos de reloj de FPGA entraron entre cada flanco de subida.

Puesto que el FPGA trabaja con un reloj de 50MHz y debido a que se usa una máquina de estados resulta que $\Delta t(s)$ es un múltiplo de 40ns por lo que la máxima velocidad a medirse podría ser

$$\omega = \frac{1rev}{400 \cdot 40 \cdot 10^{-9} x} \frac{60s}{1min}$$

$$\omega = \frac{7500000}{x} \tag{3.3}$$

Donde x es un registro que almacena el número de pulsos de reloj de FPGA entre dos flancos de subida.

Y dado que la máxima velocidad es de 3000 RPM se tiene que x siempre será mayor que 2500.

Nótese que este esquema podría medir velocidades tan pequeñas como lo permitan los bits usados para el contador, por ejemplo para un contador de 24 bits tendríamos que la mínima velocidad que podría registrase es de:

$$\omega = \frac{7500000}{2^{24}-1} = 0.44RPM \qquad (3.4)$$

Sin embargo a pesar de estas ventajas respecto al esquema de contar pulsos no se obtuvieron los resultados esperados tal como se muestra en la figura 3.8 en donde se observa que la velocidad cambia abruptamente en un tiempo pequeño lo cual no es posible.

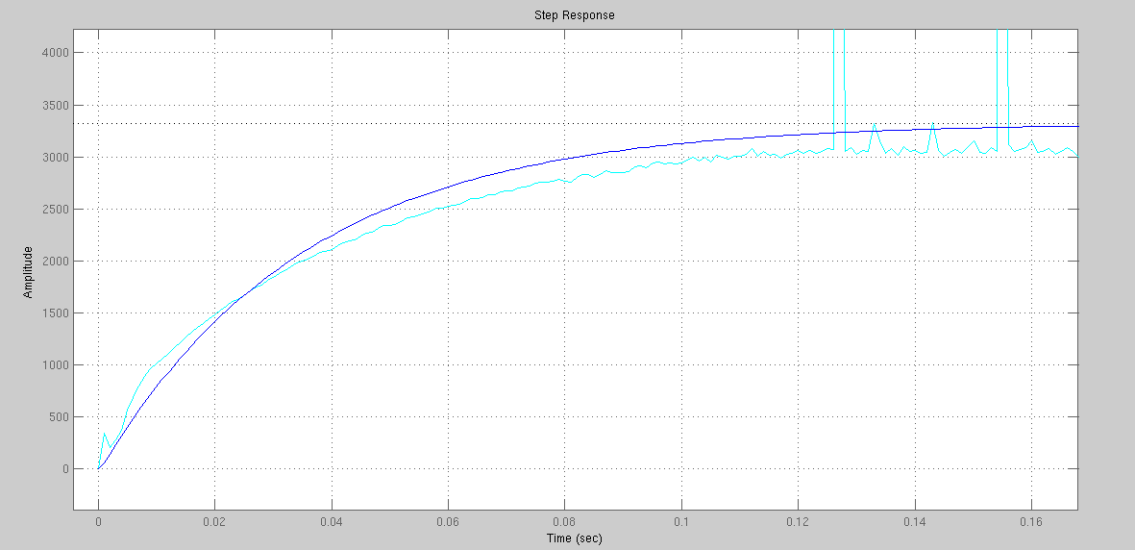


Fig.3.8 Resultados del esquema intervalo entre flancos de subida

Una posible razón del fallo de este esquema es la suposición de que los agujeros del disco del encoder se encuentran angularmente equidistantes, lo cual no es asegurado por el fabricante de manera que pueden haber mayor o menor espaciamiento entre agujeros lo cual se traduce en aparentes y abruptas subidas y bajadas de velocidad.

Así también, debe notarse que el esquema de medición de intervalo entre pulsos tiene una forma inversamente proporcional, por lo que un pequeño error en la lectura de x se traduce en un gran error en la adquisición de la velocidad angular, sobre todo cuando nos encontramos midiendo velocidades superiores a 500 RPM tal como lo muestra la siguiente figura 3.9.

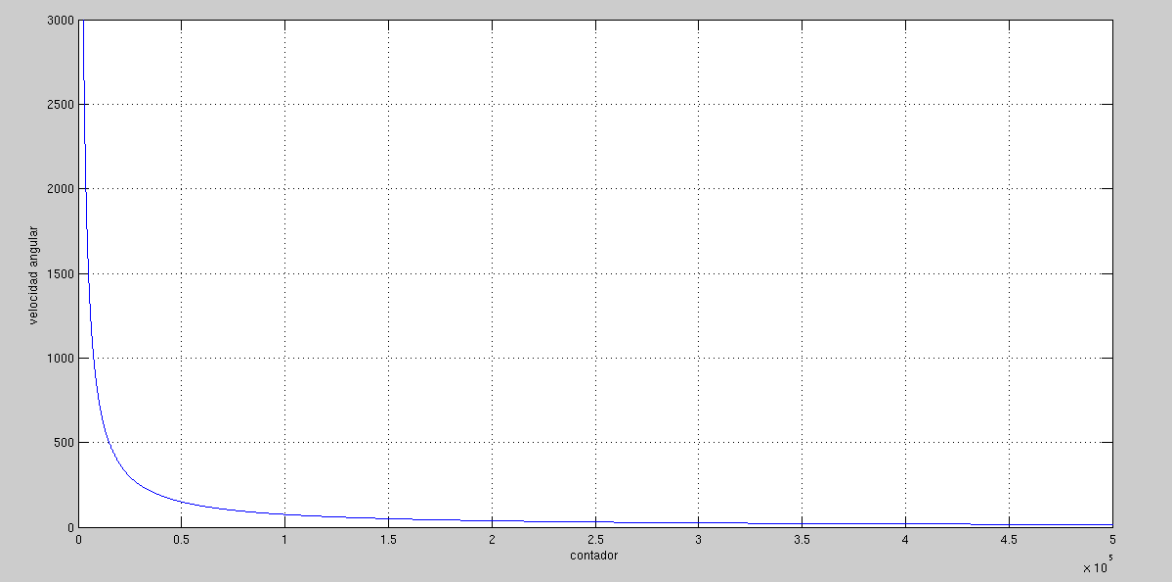


Fig.3.9 Curva velocidad vs variable contador

Dado estos inconvenientes la posibilidad de trabajar con este esquema se descartó y se siguió con el esquema de contar pulsos.

b) Esquema de cuenta de pulsos

Para el esquema de cuenta de pulsos en ventana de tiempo con intervalo de muestreo de 10ms y 400 pulsos por revolución y un motor de 3000 RPM tenemos;

$$\omega = \frac{\#pulsos}{10ms}\frac{1rev}{400\,pulsos}\frac{1000ms}{1s}\frac{60s}{1min}$$

$$\omega = (\#pulsos)(15) \qquad (3.5)$$

De manera que el factor de conversión en este caso es 15, nótese que en general el factor que relaciona el dato que enviará el FPGA con la velocidad del eje del motor es:

$$factor = \frac{1}{ppr}\frac{1}{Ts(s)} \qquad (3.6)$$

Donde:

ppr, es el número de orificios que tiene el disco del encoder, es decir, la cantidad de pulsos por revolución.

Ts, es el tiempo de muestreo en segundos que será usado.

Regresando a la ecuación 3.6 se deduce que la mínima velocidad que se puede medir es de 15 RPM y por tanto el control de velocidad se puede hacer siempre y cuando la referencia sea múltiplo de 15 RPM, si se desease una referencia de 187 RPM entonces la lectura del encoder nos dará entre 180 RPM y 195 RPM.

Por tanto se deduce que para aumentar la resolución en la medida de la velocidad angular del eje del motor debe aumentarse el número de agujeros en el disco del encoder y/o el intervalo de muestreo.

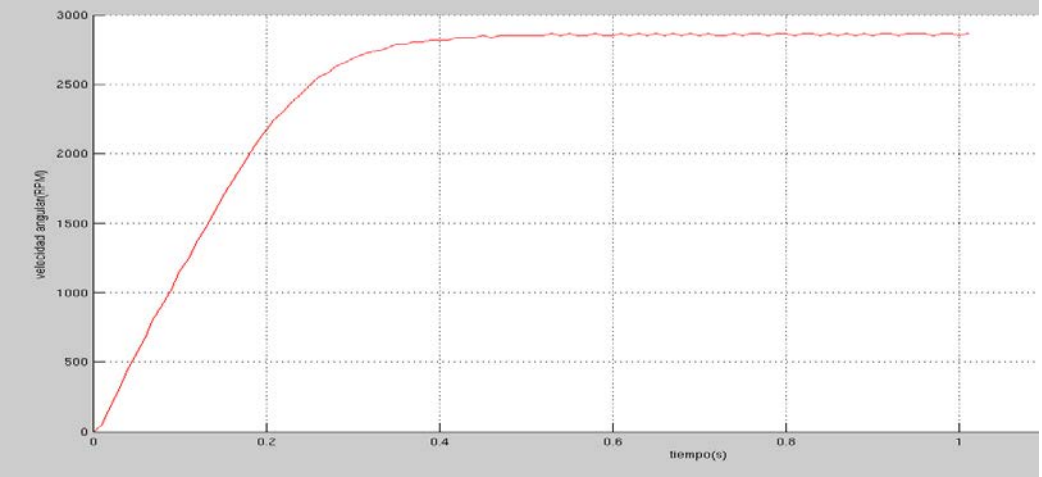


Fig.3.10 Resultados de la medición de velocidad usando el esquema de cuenta de pulsos

c) Sentido de giro

Nótese que si se sabe que el motor sólo gira en un sentido es posible determinar la posición y velocidad contando el numero de pulsos por segundo, sin embargo en un control de velocidad es necesario que el motor gire en ambos sentidos.

Los encoder disponen de dos salidas, es decir dos trenes de pulsos en cuadratura por lo que sabiendo cual esta adelantado con respecto al otro es posible determinar la posición.

Así pues se concluye en este apartado que el bloque del FPGA que procese la señal del encoder debe contar los pulsos en un intervalo de tiempo además de distinguir cual de las dos señales del encoder se encuentra adelantada respecto de la otra.

Consideremos la siguiente figura en la que canal_a se encuentra adelantada y en cuadratura respecto de canal_b, y luego a partir de un tiempo de 5us el canal_b se encuentra adelantado y en cuadratura respecto de canal_a.

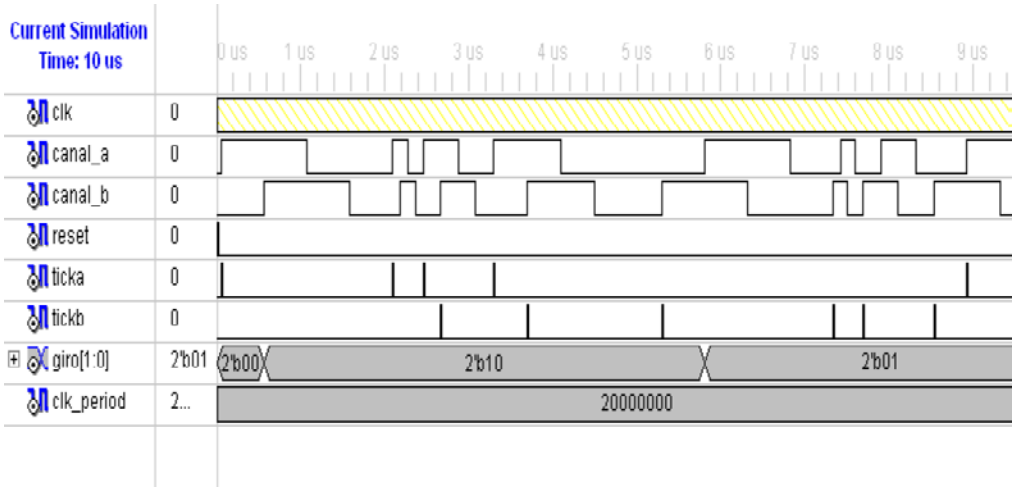


Fig.3.11 Diagrama de tiempos del proceso para obtener el sentido de giro a partir de dos señales en cuadratura

La obtención del sentido de giro se resume a procesar estas dos señales de manera que la señal giro sea “10” cuando canal_a se encuentre adelantado respecto de canal_b y sino que giro sea “01” cuando canal_b se encuentre adelantado respecto de canal_a.

La figura 3.11 es el resultado de la simulación de un módulo VHDL denominado encoderv1.0 creado en ISE 10.1 exclusivamente para probar las posibles formas de obtener el sentido de giro, el código VHDL es mostrado en el ANEXO G.

CAPÍTULO IV
IMPLEMENTACIÓN DEL SISTEMA

4.1. Comunicación y control remoto sobre protocolo UDP

Al iniciar el programa RTSJ el primer proceso que se lanza intentará enviar un comando de inicio al FPGA, sin embargo ya que la PC sólo conoce la dirección IP del FPGA más no su dirección MAC lo primero que hará es enviar un pedido ARP por lo que el FPGA debe estar en capacidad de responder a éste y luego procesar el comando de inicio.

Una vez enviado el comando de inicio la PC lanzará sus procesos tiempo real, igualmente una vez procesado el comando de inicio en el FPGA este iniciará su timer de interrupción e inmediatamente entrará a su bucle principal dándose inicio al primer periodo en el FPGA, durante el cual se hará la adquisición del primer dato $y(0)$.

A partir del segundo intervalo de muestreo en el FPGA sucederá que cada vez que éste salga de una interrupción se enviará a la PC remota la velocidad adquirida en el intervalo de muestreo anterior, es por esta razón que al comienzo del primer periodo no se puede enviar nada, ya que aunque podría enviarse un valor por defecto tal como un valor inicial nulo podría darse el caso que el motor ya este girando por algún torque aplicado y los cálculos en el comando PWM no sean los adecuados.

Nótese que el proceso de sincronización unilateral permite que el FPGA entre a su bucle principal poco después de que la PC entre en su propio bucle principal de manera que el FPGA enviará la velocidad en un momento oportuno, es decir, mucho antes de que el proceso RTSJ entre en su siguiente periodo, siempre que la distancia entre la PC y el FPGA no sea tan larga como para que las tramas demoren más de la mitad del periodo de muestreo en llegar desde la PC al FPGA y viceversa; si esto sucediese se podría solucionar el problema aumentando el periodo del programa RTSJ y del FPGA.

En la PC el proceso RTSJ esperará los datos del FPGA durante todo su intervalo de muestreo, una vez recibido un proceso tiempo real calculará el comando PWM y otro

proceso tiempo real enviará el comando momento en el cual el programa RTSJ no hará nada más que esperar por el siguiente intervalo de muestreo y repetir los procesos.

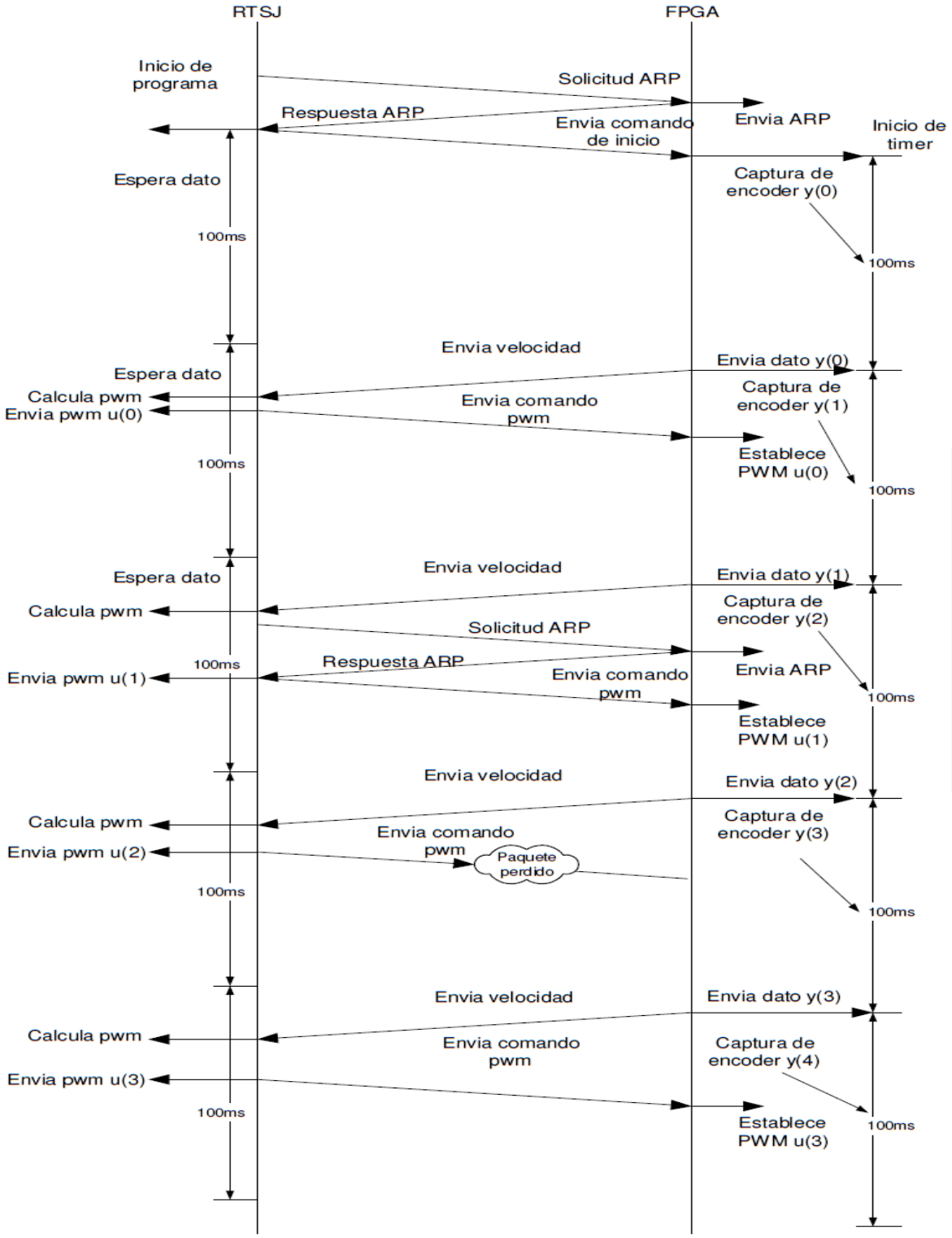


Fig.4.1 Diagrama de tiempo de la comunicación entre la PC y el FPGA (Fuente: Elaboración propia)

El FPGA luego de enviar la velocidad, esperará durante todo el intervalo de muestreo a que llegue el comando PWM, si llega antes de que se produzca la interrupción entonces se establecerá la señal PWM y luego irá al comienzo del bucle principal repitiendo otra vez los procesos, en caso de no llegar antes del pedido de interrupción de igual manera se sale del bucle secundario que se encarga de la espera del comando PWM y se irá al comienzo del bucle principal para luego enviar la velocidad.

Por lo dicho anteriormente el FPGA debe siempre enviar la velocidad al comienzo de su periodo sin importar si recibió o no el comando PWM en el periodo anterior. Esto se puede ver en la figura 4.1, en donde el FPGA espera por la señal de control $u(2)$, y a pesar de que esta no llega se envía en el comienzo del siguiente periodo los datos de la adquisición.

El FPGA debe estar preparado para responder a pedidos ARP aun cuando ya se envió la MAC a la PC, esto se debe a que la PC podría perder la dirección MAC (la tabla de direcciones es limpiada cada cierto tiempo) y solicitarla de nuevo, tal como sucede antes de enviar la señal de control $u(1)$ en la figura 4.1

4.1.1. Conexionado e interfaces electrónicas

El FPGA se conecta al LAN83C185 usando el estándar Media Independent Interface (MII) [21] tal como se muestra en la figura 4.2, más detalles sobre la interface incluyendo los números de los pines del FPGA aparecen en la tabla 4.1.

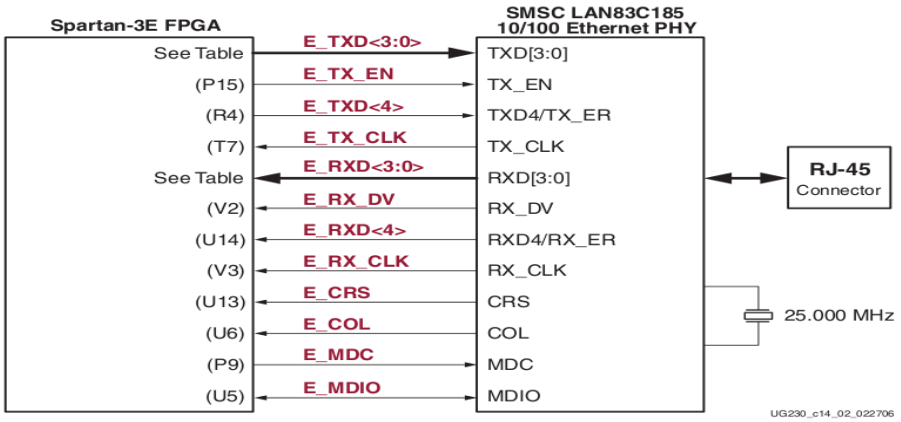


Fig.4.2 Interfaz de conexión entre el chip LAN83C185 y el FPGA Spartan3E (Fuente: Guía de usuario del kit Spartan 3E [21])

Nombre de señal	Pin en el FPGA	Función
E_TXD<4	R6	Transmisión de datos a la capa Física PHY. E_TXD<4> es también el indicador de error en la transmisión de la MII.
E_TXD<3>	T5	
E_TXD<2>	R5	
E_TXD<1>	T15	
E_TXD<0>	R11	
E_TX_EN	P15	Habilitación de transmisión.
E_TX_CLK	T7	Reloj para transmisión. 25 MHz en modo 100Base-TX, y 2.5 MHz en modo 10Base-T.
E_RXD<4>	U14	Recepción de datos desde la capa física PHY
E_RXD<3>	V14	
E_RXD<2>	U11	
E_RXD<1>	T11	
E_RXD<0>	V8	
E_RX_DV	V2	Dato recibido válido
E_RX_CLK	V3	Reloj para recepción. 25 MHz en modo 100Base-TX, y 2.5 MHz en modo 10Base-T.
E_CRS	U13	Detección de portadora
E_COL	U6	Detección de colisión en MII
E_MDC	P9	Reloj para el manejo serial de datos
E_MDIO	U5	Manejo de datos de entrada y salida

TABLA Nº 4.1 Conexión Media Independent Interface

4.1.2. Programa Java Real Time para Telecontrol

Para el control de nuestra planta (motor DC) es necesario que cada cierto tiempo (intervalo de muestreo) sea enviada la velocidad del motor desde el FPGA a la PC remota con la finalidad de que esta última realice el cálculo de la señal de control, aunque este cálculo podría realizarse en el FPGA (ya que MicroBlaze soporta operación en punto

flotante) la idea es que lo haga una PC de manera que en futuros trabajos este sistema se pueda ampliar a un sistema de control distribuido.

Dado que la PC hará un cálculo de la magnitud de la señal de control es necesario asegurar que en cada intervalo de muestreo la PC (RTSJ) reciba el dato, lo procese y envíe la señal de control adecuada de acuerdo con las especificaciones de tiempo transitorio y de estado estable.

En el FPGA tenemos implementado un microprocesador soft (MicroBlaze) con un programa descrito en C el cual no implementa un sistema operativo, es decir, el FPGA se comportará como un dispositivo en tiempo real de manera que los tiempos que demorará en procesar los datos son de naturaleza determinística.

Sin embargo la PC que controlará remotamente al motor tiene implementado un Sistema Operativo (Ubuntu 10.10) el cual no es un RTOS de manera que no nos asegura responder a un evento en un tiempo determinado, si no que el tiempo en responder dependerá de su carga de procesos.

Es por esta razón que necesariamente se tiene que hacer uso de la especificación tiempo real para java y su respectiva implementación rtsj-ri-1.5 proporcionada gratuitamente por la empresa TimeSys.

El programa RTSJ al que denominaremos *TelecontrolRTSJ* debe disponer de una clase principal que convocará otras clases y las inicializará (Builder.java), una clase periódica (ComunicacionUDP.java) que creará dos socket, uno para envío de datos y otro socket para recepción de datos, así se requiere de una clase aperiódica (Controlador.java) que hará el cálculo de la señal de control haciendo uso de un SetPoint establecido por ObtencionSP.java y la clase Matriz.java para las operaciones con matrices tal como se muestra en la figura 4.3.

En resumen el programa esperará por la velocidad del motor, procesará la velocidad para obtener la señal de control y cerrará el ciclo con el envío de esta última señal, adicionalmente el programa RTSJ enviará a un programa java la velocidad del motor para que sea graficada en una ventana.

Las clases y método a continuación vistos pueden ser encontrados en el anexo A.

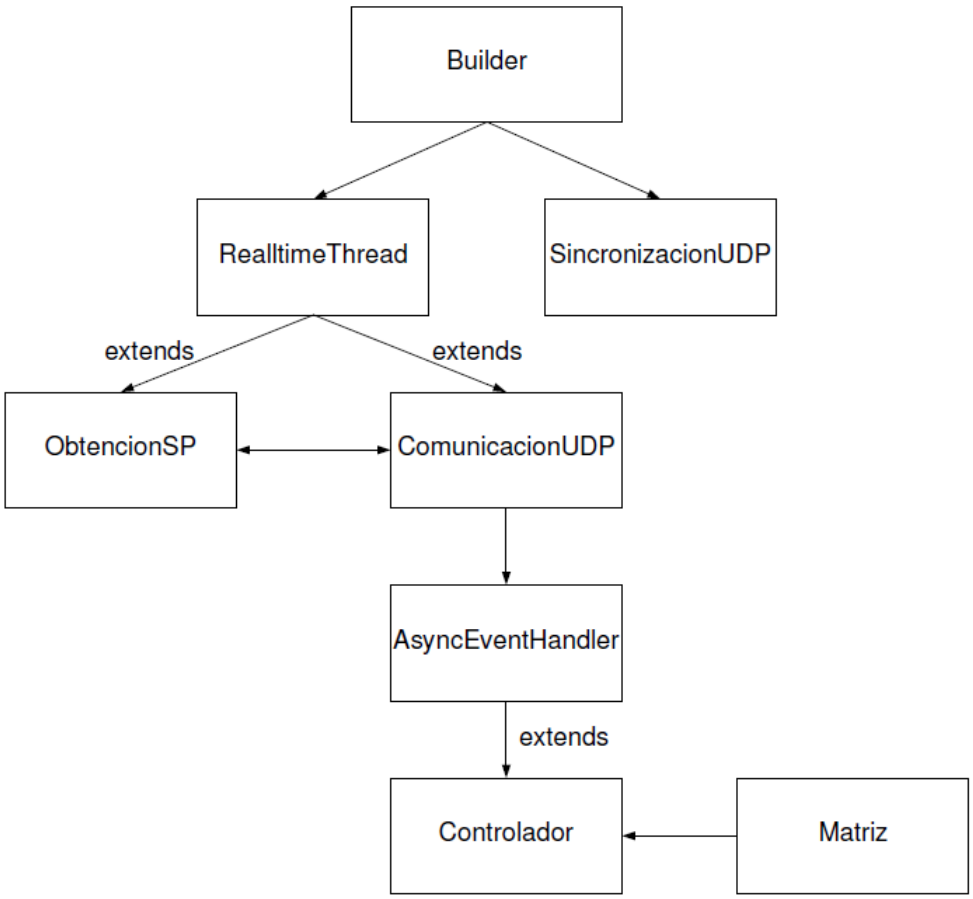


Fig.4.3 Jerarquía del programa TelecontrolRTSJ (Fuente: Elaboración propia)

a) Implementación Builder.java

La clase Builder.java cumple la función de convocar e inicializar las tareas en tiempo real, para esto hace uso del paquete javax.realtime que contiene los métodos necesarios para implementar las tareas en tiempo real.

Esta clase crea un objeto llamado *planificador* y lo establece como planificador de prioridades, este tipo de objetos, que desciende de la clase Scheduler contiene los métodos para el análisis de viabilidad, control de admisión, control de despacho y el mecanismo de admisión de gestión de eventos asíncronos, así también debe decirse que sólo puede existir uno por cada JVM (Java Virtual Machine) que esté ejecutándose.

b) Implementación ComunicacionUDP.java

Esta clase, cuyo código está basado en [26], es la encargada de la comunicación UDP, gestiona tanto la recepción de datos UDP (velocidad del motor) como el envío de datos UDP (señal de control), así también se encarga de la comunicación entre los objetos *comunicación* y *obtención*.

Esta clase hace uso de los paquetes javax.realtime para las tareas tiempo real, el paquete java.net que brinda los métodos necesarios para la comunicación UDP y el paquete java.io que brinda los métodos necesarios para la salida en pantalla de los mensajes de excepciones.

En esta clase se encuentran dos parámetros importantes tal como se muestra en la figura 4.4; el primero es la dirección IP destino, es decir, la dirección IP que tiene el FPGA y que debe ser establecido en tiempo de compilación; el otro parámetro es un factor que al multiplicar con el dato enviado por el FPGA se obtiene como producto la velocidad del eje del motor DC, este parámetro debe ser calculado y establecido en tiempo de compilación para lo cual puede verse la ecuación 3.6.

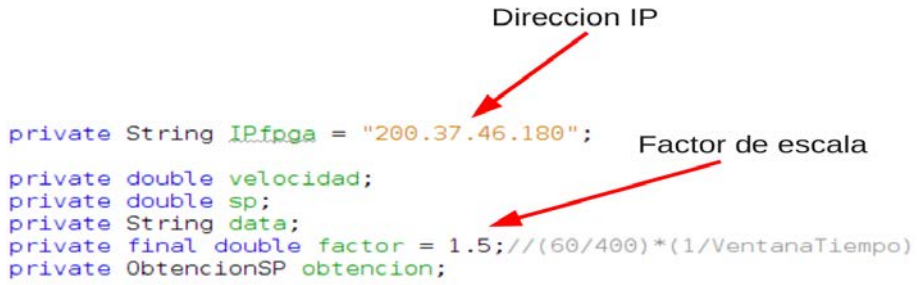


Fig.4.4 Parámetros de red y factor de escala

Debe mencionarse que esta clase de tipo periódica será la encargada de disparar el proceso aperiódico creado por la clase Controlador.java destinado al cálculo de la señal de control.

Métodos de la clase:

public void run()

Este es el método principal de la clase, su definición puede encontrarse en su definición puede encontrarse en [27], y es el encargado de recibir y procesar la trama UDP, así también, dispara un evento asíncrono para indicarle a un objeto de la clase *controlador* que la salida de la planta (velocidad) está lista para ser procesada por el algoritmo de control.

public synchronized void EnvioDato(int pwm)

Este método es disparado por un objeto de la clase *controlador*, y es el encargado de enviar la trama UDP que contiene el comando PWM que debe establecerse en el FPGA.

public synchronized void EnvioGrafico()

Este método es disparado por un objeto de la clase *controlador*, y es el encargado de enviar la trama UDP al programa Java que hace la representación gráfica de la salida de la planta (velocidad).

public double ObtenerDato()

Este método, el cual es usado por un objeto de la clase *controlador*, devuelve como dato la salida de la planta (velocidad del motor).

public double ObtenerSP()

Este método, el cual es usado por un objeto de la clase *controlador*, devuelve como dato el set point ingresado por el teclado.

c) Implementación de Controlador.java

Esta clase, cuyo código está basado en [26], recibe la salida del sistema y el set point que se requiere, a partir de estos dos datos calcula la señal de control que será enviada por protocolo UDP.

Así también esta clase mediante el método registro almacena en un archivo salida.dat la señal de control calculada y la salida del sistema que se ha recibido.

Métodos de la clase

public void handleAsyncEvent()

Este es el método principal de la clase, su definición puede encontrarse en [27], y es disparado por un objeto de la clase *comunicación*, y tiene como finalidad llamar en orden a una serie de métodos para: la observación de estados, obtención de la salida de la planta, cálculo de la señal de control, enviar dato mediante UDP al FPGA, enviar dato mediante UDP al programa que grafica la salida y guardar los datos de la señal de control y salida de la planta en un archivo con extensión .dat.

private synchronized void Observador1()

Este método, que contiene las matrices necesarias para la observación de estados, realiza el proceso para observar las variables de estados.

Debe mencionarse que si se desease agregar algún otro observador para un motor DC de características diferentes, basta con “copiar, pegar y renombrar” este método (por ejemplo renombrar a Observador2) y cambiar solo las matrices para la observación de estados. Finalmente para que este método sea usado basta con seleccionarlo en el método principal handleAsyncEvent.

private synchronized void Modelamiento1()

Este método es útil cuando se requiere hacer la identificación de parámetros de un motor DC y para su usarlo basta con seleccionarlo en el método handleAsyncEvent.

private synchronized void Controlador1()

Este método realiza el cálculo del comando PWM haciendo uso de un algoritmo de control basado en un sistema con entrada de referencia, las matrices y operaciones allí realizadas se basan en los cálculos realizados en 5.2.

private synchronized void Controlador2()

Este método realiza el cálculo del comando PWM haciendo uso de un algoritmo de control basado en un sistema de seguimiento, las matrices y operaciones allí realizadas se basan en los cálculos realizados en 5.3.

private synchronized void Registro()

Este método tiene la función de almacenar en un archivo “salida.dat” la señal de control y la salida de la planta por cada intervalo de muestreo. Este archivo puede ser encontrado en la carpeta donde son compiladas las clases.

d) Implementación de ObtencionSP.java

Esta clase, cuyo código está basado en [26], tiene el objetivo de dar los métodos necesarios para leer un valor desde el teclado y luego enviárselo a un objeto de la clase “ComunicacionUDP”.

Métodos de la clase

public void run()

Este es el método principal de la clase, y tiene como función obtener los datos, relativos al set point, ingresados por el teclado.

public synchronized double ObtenerSP()

Este método el cual es usado por un objeto de la clase *Comunicacion* devuelve el dato, esto es el set point, ingresado por el teclado

e) Implementación de SincronizacionUDP.java

Esta clase tiene como objetivo sincronizar unilateralmente el inicio las operaciones realizadas en el computador y el FPGA.

La idea principal es que el programa en el FPGA no entre a su bucle principal hasta que reciba el carácter “A”, así también el programa *TelecontrolRTSJ* apenas se inicie debe enviar este carácter “A” y luego lanzar sus procesos RT, esto permite que luego de que los procesos de *TelecontrolRTSJ* sean lanzados se haga lo mismo en el FPGA.

```
1  import java.net.*;
2  import java.io.*;
3  public class SincronizacionUDP {
4
5      private static volatile byte[] buffer_rx= new byte[1024];
6      private static volatile byte[] buffer_tx = new byte[1024];
7      private String IPfpga = "200.37.46.180";
8      private String data;
9      public SincronizacionUDP(){
10
11     }
```

Fig.4.5 Inicio de clase SincronizacionUDP

El método ReconocimientoTx que no retorna nada se encarga de enviar un carácter “A” a una IP y puerto que deben ser especificados antes de la compilación.

```
13      public void ReconocimientoTx(){
14          //Envio de Señal de requerimiento al FPGA
15          try{
16              data="A";
17              buffer_tx = data.getBytes();
18              InetAddress destino = InetAddress.getByName("localhost");
19              DatagramPacket datagrama_tx =
20              new DatagramPacket(buffer_tx,buffer_tx.length,destino,4406);
21              DatagramSocket socket_tx = new DatagramSocket();
22              socket_tx.send(datagrama_tx);
23              socket_tx.close();
24          }
25          catch(IOException e){
26              System.out.println("IO Error: " + e);
27          }
28          return;
29      }
30
```

Fig.4.6 Método ReconocimientoTx

El método ReconocimientoRx espera hasta que se reciba un carácter “A”, una vez esto suceda se pone en 1 la variable *IndicadorInicio* la cual es retornada.

```
31      public int ReconocimientoRx(){
32          int IndicadorInicio=0;
33          //Recepcion de señal de reconocimiento desde la PC
34          try{
35              DatagramSocket socket = new DatagramSocket(4406);
36              DatagramPacket datagrama = new
37                      DatagramPacket(buffer_rx,buffer_rx.length);
38              socket.receive(datagrama);
39              String cadena =
40              new String(datagrama.getData(),0,datagrama.getLength());
41              if (cadena.equals("A")) IndicadorInicio = 1;
42              socket.close();
43          }
44          catch(IOException e){
45              System.out.println("IO Error: " + e);
46          }
47          return IndicadorInicio;
48      }
49
50  }
```

Fig.4.7 Método ReconocimientoRx

Los beneficios de usar esta clase pueden observarse en 6.1.1.

4.1.3. Programa Java Real Time para simulación

Este programa tiene la finalidad de simular el comportamiento del FPGA, es decir con este programa y *TelecontrolRTSJ* se podrían probar los algoritmos de control antes de usar el motor real.

El programa RTSJ al que denominaremos SimulacionFPGA debe disponer de una clase principal que convocará a otras clases y las inicializará (Builder.java), una clase periódica (EnvioUDP.java) que creará un socket para envío de datos, una clase periódica (RecepcionUDP) que creará un socket para recepción de datos, una clase aperiódica (Motor) que simulará el comportamiento del motor haciendo uso de un modelo en el espacio de estados previamente ingresado, una clase para la sincronización unilateral y la clase Matriz.java para las operaciones con matrices tal como se muestra en la figura 4.8.

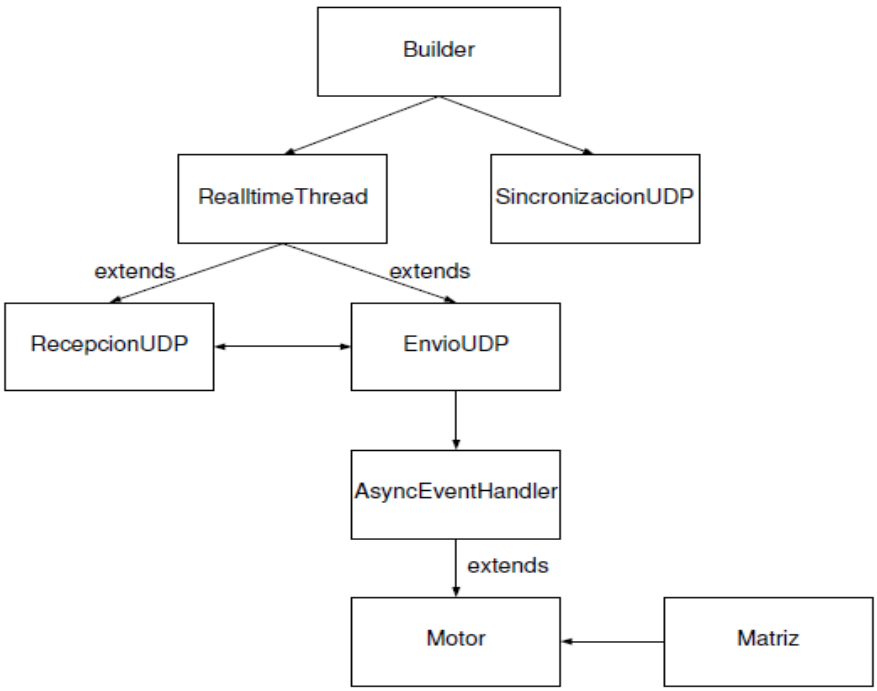


Fig.4.8 Jerarquía del programa SimulacionFPGA (Fuente: Elaboración propia)

a) Implementación Builder.java

La clase Builder.java cumple la función de convocar e inicializar las tareas en tiempo real, para esto hace uso del paquete javax.realtime que contiene los métodos necesarios para implementar las tareas en tiempo real.

Esta clase crea un objeto llamado *planificador* y lo establece como planificador de prioridades respectivamente, este tipo de objetos que desciende de la clase Scheduler contiene los métodos para el análisis de viabilidad, control de admisión, control de despacho y el mecanismo de admisión de gestión de eventos asíncronos, así también debe decirse que sólo existe uno por JVM (Java Virtual Machine).

b) Implementación EnvioUDP.java

Esta clase, cuyo código está basado en [26], es la encargada de enviar la velocidad del motor mediante UDP y darle el voltaje de control a la clase Motor.

Esta clase hace uso de los paquetes javax.realtime para las tareas tiempo real, el paquete java.net que brinda los métodos necesarios para el envío de datos mediante UDP y el paquete java.io que brinda los métodos necesarios para la salida en pantalla de excepciones.

En esta clase se debe establecer la dirección IP destino, es decir, la dirección a donde serán enviados los datos, en el caso que este programa sea ejecutado en la misma máquina que se ejecuta *TelecontrolRTSJ* se debe establecer como IP la dirección localhost tal como se muestra en la figura 4.9

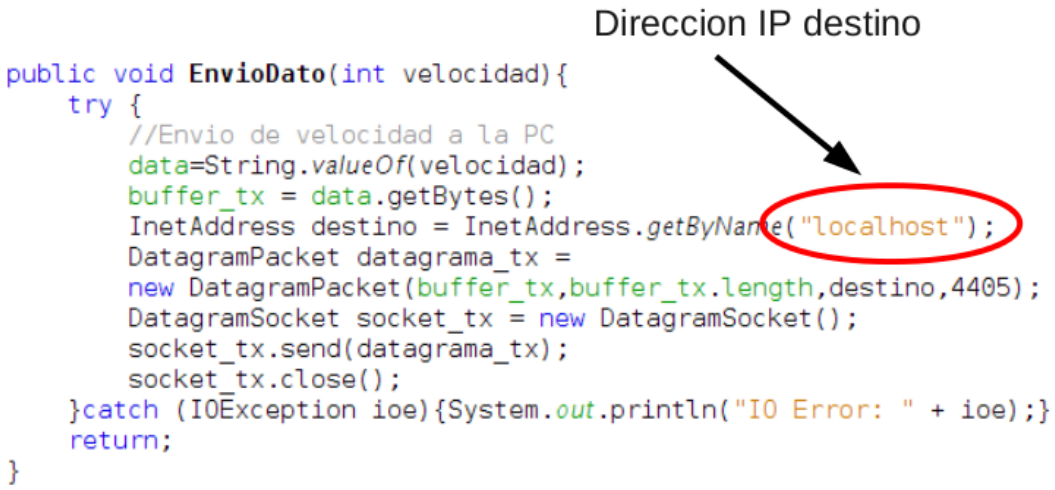


Fig.4.9 Dirección IP destino

Debe mencionarse que esta clase de tipo periódica será la encargada de disparar el proceso aperiódico creado por la clase Motor.java destinado a la simulación del motor.

c) Implementación de Motor.java

Esta clase, cuyo código está basado en [26], recibe el voltaje de control y hace el cálculo de la velocidad que tendría el motor según el modelo ingresado.

```
//Matrices de estado
private double[][] A={{-0.096401,-4.802148},{0.0250279,0.596961}};
private double[][] B={{0.0250279},{0.00210056}};
private double[][] C= {{0,18350}};
```

Fig.4.10 Modelo del motor en el espacio de estados

```
public void handleAsyncEvent() {
    //Observador de Estados
    u[0][0]= envio.ObtenerDato();//en RPM
    Matriz.Mult(A, 2, 2, x, 2, 1, Ax);
    Matriz.Mult(B, 2, 1, u, 1, 1, Bu);
    Matriz.Suma(Ax,2, 1,Bu,2, 1,x);
    Matriz.Mult(C, 1, 2, x, 2, 1,y);
    velocidad=(int)(0.6667*y[0][0]);
    envio.EnvioDato(velocidad);
    System.out.println("u="+u[0][0]+" y="+velocidad*1.5);
    envio.schedulePeriodic();
}
```

Fig.4.11 Método principal handleAsyncEvent de la clase Motor.java

d) Implementación de RecepcionUDP.java

Esta clase, cuyo código está basado en [26], tiene el objetivo de dar los métodos necesarios para leer un valor desde el teclado y luego enviárselo a un objeto de la clase "EnvioUDP".

```
public void run(){
   RealtimeThread hiloactual = currentRealtimeThread();
   while(true){
       do{
           try {
           DatagramPacket datagrama_rx =
           new DatagramPacket(buffer_rx, buffer_rx.length);
           DatagramSocket socket_rx = new DatagramSocket(4406);
           socket_rx.receive(datagrama_rx);
           socket_rx.close();
           String cadena =
           new String(datagrama_rx.getData(),0,datagrama_rx.getLength());
           u = factor*Integer.parseInt(cadena);//en voltios
           }
           catch (IOException ioe){
               System.out.println("IO Error: " + ioe);
           }
       }while(hiloactual.waitForNextPeriod());
   }
}
```

Fig.4.12 Método principal run de la clase RecepcionUDP.java

4.1.4. Código C para microprocesador MicroBlaze

El código C a compilar estará compuesto de una función main() tipo entero y ficheros que implementaran funciones y variables globales.

Son cuatro los ficheros desarrollados y se describen brevemente a continuación:

- El fichero interrupcion.h implementa tres funciones y dos variables globales externas con el fin de inicializar y manejar los procesos de interrupción provocados por el timer.
- El fichero estructuras.h implementa un conjunto de variables globales, estructuras y constantes que serán usadas principalmente para la comunicación UDP.
- El fichero eth_ip_udp.h implementa las funciones necesarias para la comunicación UDP.
- El fichero utilitario.h implementa tres funciones relacionadas con tareas generales y de ayuda para la comunicación UDP.

Debe mencionarse que para el desarrollo del programa en MicroBlaze se ha hecho uso del estilo de programación modular [28], lo cual facilita su compresión y reusabilidad.

a) Fichero interrupcion.h

Este fichero contiene la implementación de tres funciones y dos variables globales que cumplen con lo especificado en la documentación API para el controlador de interrupciones [16] y un timer respectivamente [17].

void manejador_interrupcion (void * baseaddr_p)

Es la función que será llamada automáticamente cada vez que se produzca una interrupción provocada por el timer.

Parámetros:

baseaddr_p, Dirección de memoria del periférico que genera la interrupción

Retorna:

Nada

Función void init_interrupcion(int x)

Es la función encargada de inicializar el periférico que maneja las interrupciones y de establecer el tamaño del contador en el timer.

Parámetros:

x, un número entero de 32 bits que representara el intervalo de tiempo en milisegundos que se quiere contar.

Retorna:

Nada

void iniciar_timer (void)

Esta función inicializa la cuenta del timer además de habilitar la generación del timer y habilitar las interrupciones en procesador.

Parámetros:

Ninguno

Retorna:

Nada

b) Fichero estructuras.h

Este fichero posee declaraciones de variables globales, definiciones de estructuras y constantes que serán usadas por el fichero eth_ip_udp.h

Es en este fichero que se define la MAC Ethernet, la dirección IP y el puerto UDP que será asignado al FPGA, así también posee la MAC Ethernet, la dirección IP y el puerto UDP de la PC remota los cuales deben tener los valores adecuados.

Así también en este fichero se definen cuatros estructuras de acuerdo con los encabezados de los distintos protocolos usados.

Estructura trama_eth:

Donde:

mac_dst[6]: representa la dirección MAC destino

mac_org[6]: representa la dirección MAC origen

type/length: representa los 2 bytes de este campo

Estructura trama_ip

Donde:

v_hl_tos: representa

len: representa

id: representa

offset: representa

ttl_proto: representa

chksum: representa

ip_org[4]: representa

ip_dst[4]: representa

Estructura trama_udp

Donde:

udp_org: representa

udp_dst: representa

udp_len: representa

udp_chksumt: representa

Estructura trama_arp

Donde:

Hw_type: que para una red Ethernet debe establecerse como 0x0001

Prot_type: que para el protocolo IP es 0x0800

Hw_size: que representa el tamaño en bytes usado para una dirección MAC

Prot_size: igual que hw_size con la diferencia que es para una dirección IP

opcode: es un código de operación, si es un requerimiento de MAC debe ser 0x0001, si es una respuesta con la MAC debe ser 0x0002

mac_org: representa la dirección MAC del dispositivo que envía la trama ethernet

ip_org: representa la dirección IP del dispositivo que envía la trama Ethernet

mac_dst: representa la dirección MAC del dispositivo que debe recibir la trama Ethernet

ip_dst: representa la dirección IP del dispositivo que debe recibir la trama Ethernet

Finalmente en este fichero también se encontrarán algunas constantes que definen, por ejemplo, el tamaño de una cabecera de un determinado protocolo y tamaños de determinados buffer de memoria.

c) Fichero eth_ip_udp.h

Este fichero declara una variable global externa e implementa cuatro funciones las cuales cumplen con lo especificado en la documentación API [15] (Application Program Interface) de la IP core XPS EMACLITE.

La variable global externa Data2Pwm compartida por el fichero principal y este para almacenar un dato de 32bits que debe establecerse en la IP que genera las señales PWM.

Debe mencionarse que parte del código de este fichero está basado en [29] y [30].

void inicializacion_xemaclite(XEmacLite *EmacPtr)

Esta función es la encargada de inicializar el periférico XEmacLite Ethernet, de manera que inicializa el periférico, establece la dirección MAC y limpia el buffer de recepción de tramas ethernet.

Parámetros:

EmacPtr, Dirección de memoria del periférico XEmacLite Ethernet

Retorna:

Nada

Xuint16 respuesta_arp(Xuint8 *buf)

Esta función construye la trama ARP que debe ser enviada cada que se reciba un pedido de la dirección MAC. La construcción de la trama es hecha con los datos contenidos en el fichero estructuras.

Parámetros:

buf, un puntero a una dirección de memoria que servirá como buffer de transmisión.

Retorna:

La suma de los tamaños de las cabeceras Ethernet y ARP

int construccion_trama(Xuint8 *buf_msje, int long_msje, Xuint8 *buf)

Esta función construye la trama Ethernet, IP, UDP y de datos.

Parámetros:

buf_msje, un puntero a una dirección de memoria estructurada en bytes que tiene los datos que quiere enviarse en la capa de aplicación.

long_msje, es un entero cuyo valor debe representar el tamaño en bytes del buffer que contiene el mensaje.

buf, debe contener un puntero a la dirección a un buffer de memoria usado sólo para almacenar la trama Ethernet que se quiere transmistir.

Retorna:

El tamaño en bytes de la trama Ethernet incluyendo la cabecera Ethernet, IP, UDP y los datos de la capa de aplicación.

Xuint8 analisis_trama(Xuint8 *buf)

Esta función analiza una trama Ethernet recibida y retorna distintos valores dependiendo de la naturaleza de la trama.

Parámetros:

buf, debe contener un puntero a la dirección a un buffer de memoria usado sólo para almacenar la trama Ethernet que se ha recibido.

Retorna:

Un byte que dependiendo de la naturaleza de la trama tiene los siguientes valores:

0x01 si es un pedido de la dirección MAC en ARP.

0x02 si la siguiente capa no pertenece al protocolo IP.

0x03 si la dirección MAC no coincide con la MAC del FPGA

0x04 si la capa siguiente a la capa IP no pertenece al protocolo UDP

0x05 si se ha recibido el valor PWM a establecer

0x06 si se ha recibido el carácter A en ASCII (0x41)

d) Fichero utilitario.h

Este fichero contiene un buffer de memoria de 9 bytes para la comunicación con el fichero principal y tres funciones útiles al fichero eth_ip_udp.h

La variable global externa d[9] es compartida por el fichero principal y este para almacenar 9 bytes que serán los datos de la capa de aplicación que se quiere transmitir.

void copiar_datos(void *org, void *dst, int n)

Esta es una función que simplifica la labor de hacer la copia de un buffer memoria o arreglo a otro buffer de memoria o arreglo, recibe como parámetros un puntero al buffer o arreglo del cual se quiere hacer la copia, un puntero a un buffer o arreglo en el cual se copiara los datos y un entero con la longitud del arreglo o buffer.

Parámetros:

org, un puntero a una dirección de memoria que contiene datos que se quieren copiar.

dst, un puntero a una dirección de memoria al cual se quieren copiar los datos

n, un entero cuyo valor es el numero de datos que se quiere copiar

Retorna:

Nada

void dato2string(unsigned long x)

Esta función recibe un dato de 32 bits proveniente de la IP de adquisición de velocidad. Este dato contiene información del sentido y magnitud de la velocidad angular procesado a partir de dos señales de un encoder.

Parámetros:

x, un número de 32 bits el cual contiene información del sentido y magnitud de la velocidad calculado en la IP core de adquisición.

Retorna:

Nada

Xuint16 chksum16(void *buf, short longitud)

Esta función obtiene la suma de cabecera según [9].

Parámetros:

buf, es un puntero al origen de la trama IP en algún buffer de memoria de la cual se quiere calcular la suma de verificación.

longitud, es el número en bytes de la longitud de una cabecera IP.

Retorna:

La suma de verificación de 16 bits.

e) Fichero main.c

Este fichero es el programa principal y su diagrama de flujo se muestra en la figura 4.13, el uso de los modificadores, directivas del preprocesador y el estilo de programación basado en la arquitectura de bucle infinito con interrupciones para sistemas embebidos puede ser encontrado en [31]

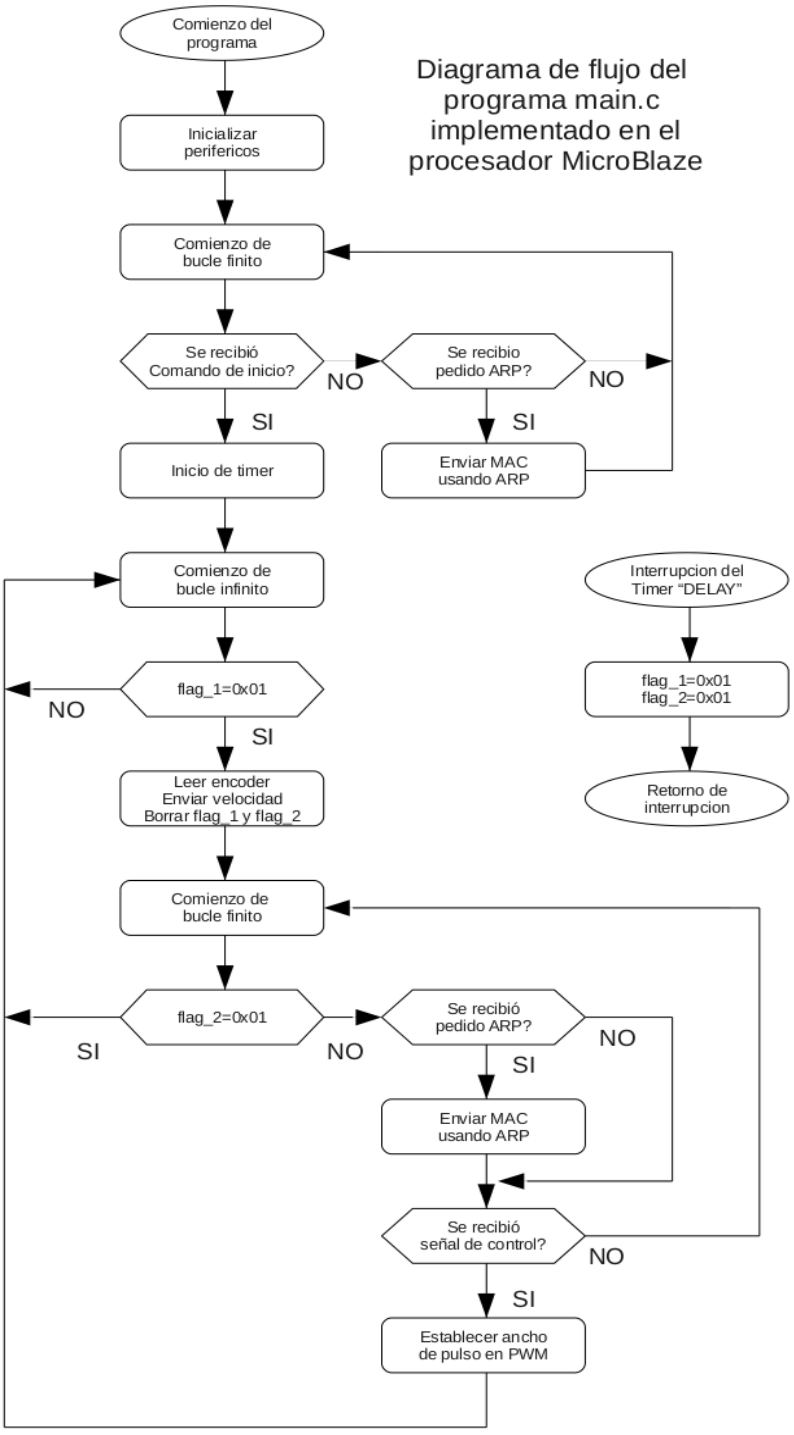


Fig.4.13 Diagrama de flujo del programa en C implementado en el procesador MicroBlaze

4.2. Concepción de los periféricos para el control de un motor DC

4.2.1. Modulador por ancho de pulso

a) Diseño de IP para procesador MicroBlaze

Esta IP core genera dos señales PWM y una señal de habilitación las cuales son dirigidas a tres pines de salida del FPGA para controlar la etapa de potencia. Para este fin es usado un bloque denominado base de tiempo, un bloque Contador, un bloque comparador, un bloque llamado Driver, un bloque denominado Dead Time y por último un bloque denominado Comunicación con IPIF.

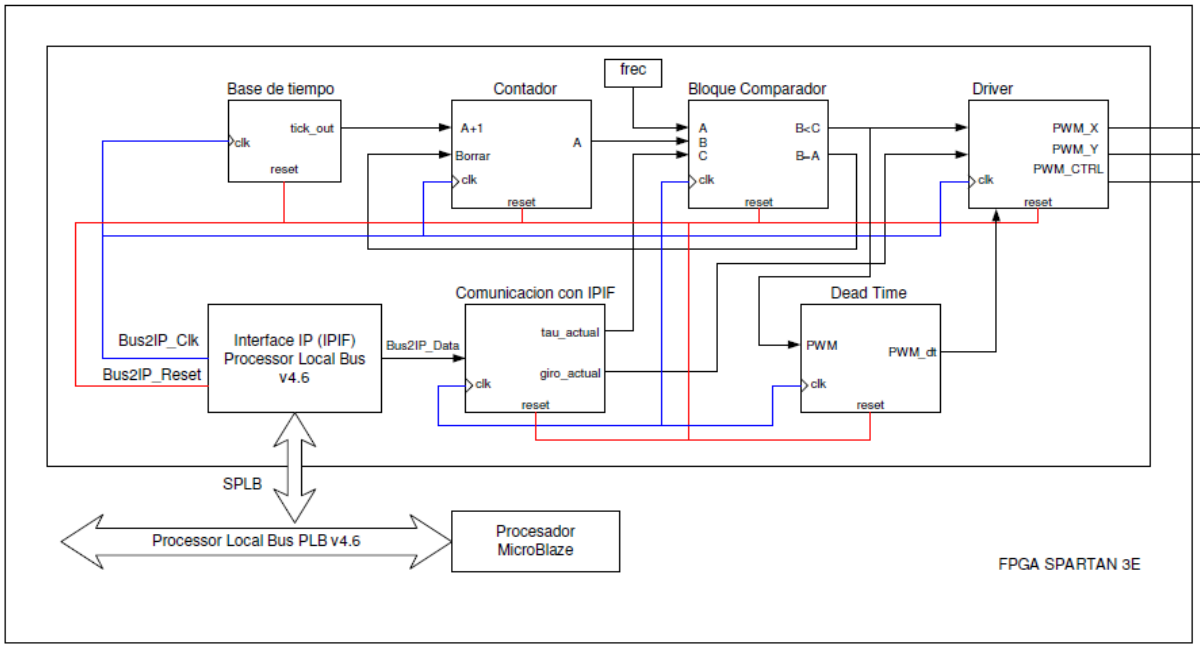


Fig.4.14 Diagrama de bloques del módulo PWM (Fuente: Elaboración propia)

El bloque base de tiempo permite establecer la resolución de la señal PWM, este envía al bloque contador un pulso de reloj cada cierto tiempo para que aumente en uno el valor de su registro.

El bloque contador está basado en un registro que aumenta en uno el valor contenido en el cada vez que su entrada “A+1” sea activada, así también dispone de la entrada “borrar” la cual borra el contenido del registro cada vez que es activada.

El bloque comparador está compuesto por dos comparadores, un comparador activa su salida cuando el contenido del registro contador es igual al valor de un parámetro denominado “frec” el cual establece la frecuencia de la señal PWM.

El otro comparador tiene activa su salida siempre que el contenido del registro contador sea menor que el valor de un registro denominado “tau”, el cual controla el ancho del pulso de la señal PWM, nótese que la salida de este comparador posee la señal PWM.

El bloque comunicación con IPIF tiene como función recibir los datos que envía el procesador y almacenar en un registro el ancho de pulso que debe establecerse en la señal PWM, así también almacena en un flip-flop la información sobre qué salida debe contener la señal PWM.

El bloque “driver” recibe la señal PWM y tiene como función determinar qué salida deberá contener a esta.

Bloque base de tiempo

Este bloque está basado en un registro y un sumador, este registro con ayuda del sumador aumenta su valor con cada flanco de subida del reloj del FPGA, una vez que el registro alcance un valor determinado este bloque activara su salida, es decir, su salida tick_out presentará un pulso activo alto cuya duración coincidirá con el periodo de reloj del FPGA.

El valor que debe alcanzar el registro para activar la salida de este bloque está determinado por un parámetro definido en tiempo de compilación, este parámetro es denominado M y se encuentra en la sección correspondiente a los parámetros genéricos de la entidad VHDL, así también se encontrará un parámetro denominado N el cual define el tamaño en bits del registro que realizará la cuenta, nótese que N es el máximo entero del logaritmo de M en base 2.

A manera de ejemplo considerando el código VHDL presentado en el Anexo E, tenemos que si M=50 entonces la resolución de la señal PWM será de 1 us, es decir el mínimo tiempo en alto que puede estar la señal PWM es de 20 us, es por eso que el bloque anterior se le ha denominado base de tiempo para resolución PWM.

Así también debe decirse que el FPGA trabajando a 50 MHz nos podría dar una resolución PWM de hasta 20 ns, sin embargo esto sería un desperdicio de recursos puesto que ningún puente H así sea basado en MOSFET puede conmutar a tal velocidad, Es así que viendo la hoja de datos del integrado L298 se eligió que la resolución de PWM sea de 1 us con un periodo de 10 ms, lo que nos da un actuador de $10\text{ms}/1\text{us} = 10000$, esto es de 9 bits.

Debe agregarse que siendo la resolución y la frecuencia dos magnitudes dependientes de dos parámetros genéricos sólo basta cambiar sus valores en tiempo de compilación si se desease mayor o menor resolución según el actuador.

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

Salidas:

tick_out, esta señal presenta un pulso de reloj activo alto cada vez que la cuenta llega a un valor determinado.

Bloque contador

Este bloque está basado en un registro y un sumador de 16 bits, este registro con ayuda del sumador aumenta su valor cada vez que la entrada "A+1" presente un pulso activo alto proporcionado por el bloque base de tiempo, de manera que el reloj del FPGA sirve sólo para actualizar los registros mas no para aumentar la cuenta que realiza este bloque.

A la salida de este bloque tendremos un dato de 16 bits el cual representara el valor del registro, al comparar este dato con la salida de otros registros se obtendrá la señal PWM y una señal denominada reiniciar la cual indicara el momento en que debe limpiarse el contenido del registro de este bloque.

En resumen cuando reiniciar sea activo alto este registro debe comenzar en cero, cuando su entrada A+1 sea activo alto el registro debe incrementarse en uno y de no ocurrir ninguno de estos entonces debe mantener su valor.

Nótese que del ejemplo anterior el contador se incrementaría cada 1us.

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

A+1, señal que incrementa la cuenta en uno cada vez que se activa con pulso de reloj activo alto

Borrar, señal que limpia el registro cada vez que se activa con un pulso de reloj activo alto.

Salidas:

A, salida de 16 bits con el valor actual del registro de este bloque.

Bloque comparador

Este bloque se encuentra compuesto por dos comparadores de 16 bits, estos tienen una entrada común la cual es el valor del registro contador, un comparador compara el valor de este último registro con el valor del parámetro *frec* definido como constante y que puede variarse sólo en tiempo de compilación, de manera que cuando sean iguales se genere en la salida *reiniciar* un pulso activo alto que de la orden de limpiar el registro contador, nótese que con este comparador se determina la frecuencia de la señal PWM de manera que si *frec* tiene un valor de 10000 y la resolución de PWM 1 us, entonces la señal PWM tendrá una frecuencia de 10 ms.

El otro comparador compara el valor del registro tau con el valor del registro contador, de manera que cuando este último sea menor la salida estará activo alto obteniéndose así la señal PWM.

Entradas:

A, entrada de un número de 16 bits de tipo constante

B, entrada del valor actual del registro del bloque contador

C, entrada del valor actual del registro que contiene el ancho del pulso de la señal PWM.

Salidas:

B<C, salida de 1 bit que tiene estado activo alto cuando el valor del registro contador es menor que el valor del registro tau.

B=A, salida de 1 bit que tiene estado activo alto cuando el valor del registro contador coincide con el valor constante denominado frec.

Bloque DeadTime

Este bloque genera un tiempo muerto en la señal de habilitación, del CI L298, cada vez que haya un cambio en la señal PWM.

Para este fin el bloque usa una máquina de estados cuyo diagrama de estados se muestra en 4.15.

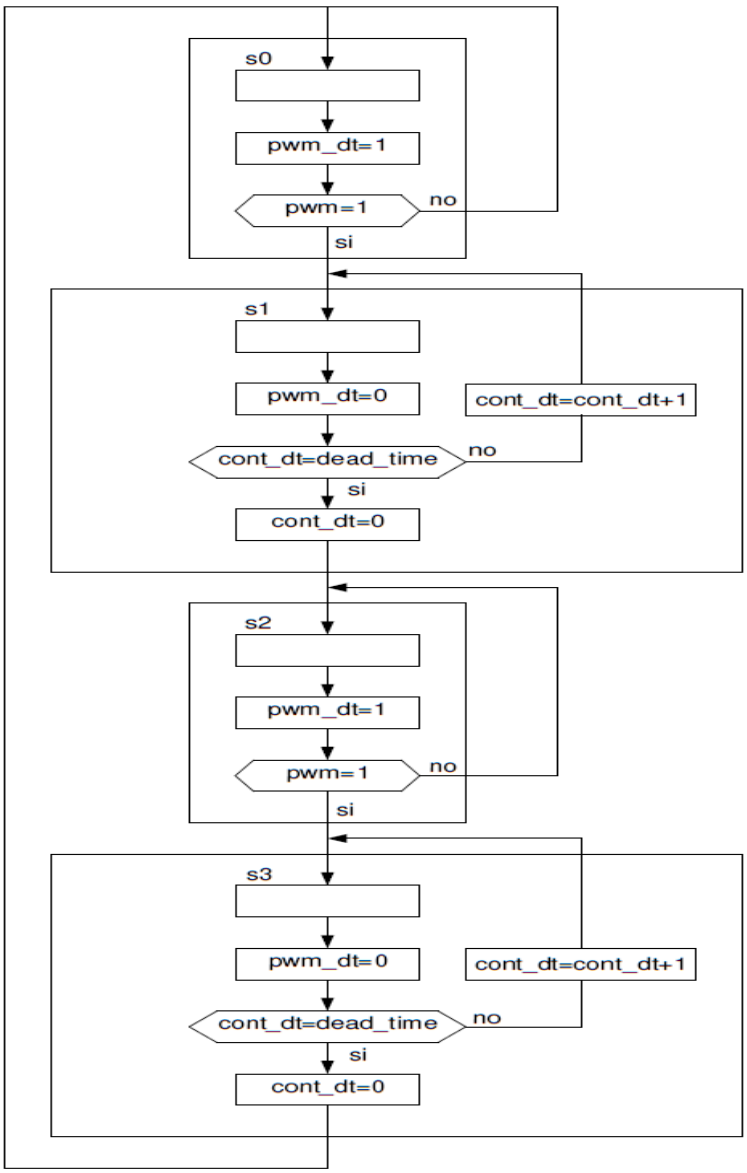


Fig.4.15 FSM para la generación del tiempo muerto (Fuente: Elaboración propia)

Bloque Comunicación con IPIF

Este bloque tiene como función capturar los datos enviados por el procesador y almacenar en registros los datos adecuados.

El procesador enviará datos de 32 bits de los cuales sólo 17 bits tienen información relevante, de estos, 16 bits representarán la magnitud de la señal PWM (ancho de pulso) y 1 bit representará el sentido de la señal PWM.

De los 32 bits que se disponen como ancho del bus, se usará al bit 15 para la definición del sentido y del bit 16 al bit 31 para la determinación del ancho del pulso.

Siendo el formato de trabajo del procesador hacer que se enumeren los bits de izquierda a derecha comenzando desde cero además de establecer como el bit más significativo el primero de la izquierda tendremos que si se desea un ancho de pulso de 5us trabajando con una resolución de 1us el procesador debería enviar 0x00000005:

Bit 0-7	Bit 8-15	Bit 16-23	Bit 24-31
0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 **15**	**16 17 18 19 20 21 22 23**	**24 25 26 27 28 29 30 31**
No en uso	Sentido	Magnitud	Magnitud
0000 0000	0000 000**0**	**0000 0000**	**0000 0101**
0x00	0x00	0x00	0x05

TABLA Nº 4.2 Formato de envío de comando PWM de procesador a IP core

Entradas:

Bus2IP_Clk, el cual es una señal de reloj de 50 MHz

Bus2IP_Reset, el cual es una señal de reset asíncrono

Bus2IP_data, el cual es un conjunto de señales que representa una palabra de 32 bits enviada por el procesador.

Salidas:

Tau_actual, salida de 16 bits con el valor actual del registro tau que contiene el valor del ancho del pulso que debe tener la señal PWM.

Giro_actual, salida de 1 bit que contiene la información sobre el sentido de la señal PWM.

Bloque Driver

Este bloque determina que salida deberá contener la señal PWM, si la variable giro tiene como valor 0 entonces la señal PWM lo tendrá la salida PWM_X, en caso contrario será la salida PWM_Y.

En este bloque también se puede cambiar la configuración de la señal PWM en el sentido de que sea activo alto o activo bajo, esto puede ser de utilidad en caso se trabaje con distinto módulos de potencia en donde se tenga que alternar entre lógica de estado activo alto o lógica de estado activo bajo. En el anexo E es mostrado el código cuando es configurado para lógica de estado activo bajo.

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

Giro_actual, señal que contiene la información sobre el sentido de la señal PWM

Pwm_sgte, señal que contiene la salida PWM con el ancho de pulso requerido

Salidas:

PWM_X, salida que contiene la señal PWM según contenido de giro_actual.

PWM_Y, salida que contiene la señal PWM según contenido de giro_actual.

PWM_CTRL, salida que habilita o deshabilita el C.I. L298.

Nota: Es necesario agregar que si se desea cambiar la resolución de la señal PWM se debe variar N, M y *frec*, por ejemplo, si se quisiera cambiar la resolución de 1us a 100us manteniendo el periodo de 10ms entonces se debe cambiar M de 50 a 500, N de 8 a 9 y *frec* de 10000 a 1000.

El código VHDL de esta IP, así como un resumen de la síntesis del circuito se encuentra en el anexo E.

b) Interfaz de potencia

La etapa de potencia está basada en tecnología BJT usada por el C.I. L298, este C.I. posee cuatro transistores que funcionan en estado de corte y saturación.

El fabricante de este C.I. muestra en su hoja de datos distintas recomendaciones tales como:

- Adición de 4 diodos de trr menor a 120 ns.
- Adición de un capacitor que mantenga estable la alimentación de la etapa lógica (5 V).
- Adición de un capacitor que mantenga estable la alimentación de la etapa de potencia (24 V)

Siguiendo estas recomendaciones se opto por:

- Adición de 4 diodos de código FMPG12 con un trr máximo de 80ns
- Adición de un capacitor de 100nF ubicado a 1cm del C.I.
- Adición de dos capacitores, uno de 100 nF y otro de 470uF ubicados a 3cm y 1cm del C.I. para filtrar ruido de alta y baja frecuencia respectivamente.

Adicionalmente se consideró conveniente aislar ópticamente las conexiones entre el FPGA y la etapa de potencia. Para esto se añadió tres optoacopladores basados en un inversor Schmitt por lo que sus salidas son compatibles con lógica TTL además de trabajar hasta una frecuencia de 1MHz. Adicionalmente este módulo tiene los dispositivos

necesarios para que en futuros trabajos se haga la adquisición de la corriente de armadura del motor.

El circuito de potencia se muestra en la figura 4.14.

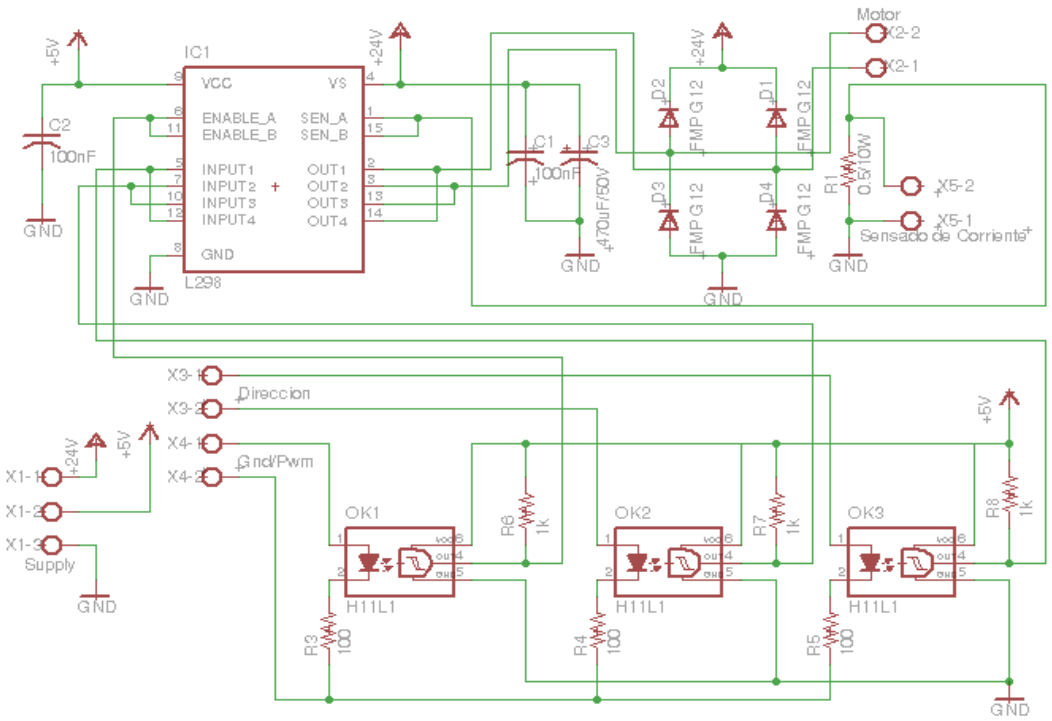


Fig.4.16 Diagrama del circuito electrónico de potencia

c) Conexionado de circuito

La etapa de potencia se conecta con el FPGA a través de tres optoacopladores, la conexión de tierra del FPGA es determinada por la conexión en la que convergen los cátodos de los tres fotodiodos, la señal de habilitación del C.I. es transferida del FPGA al L298 por medio del optoacoplador OK1, nótese que estas dos señales tiene como soporte físico la bornera X4.

La señal PWM_X toma el pin E8 del FPGA y es transferida al C.I. L298 por medio del optoacoplador OK2, la señal PWM_Y toma el pin F8 y es transferida al C.I. L298 por medio del optoacoplador OK3, estas señales usan la bornera X3. Nótese que el FPGA no debe compartir conexión de tierra con la etapa de potencia por ningún motivo.

La bornera X2 es la salida de la etapa de potencia (actuador)

En la bornera X1 deben conectarse la alimentación de la etapa de potencia y la etapa lógica. X1-1 es la entrada para el voltaje que alimentará el motor, X1-2 es la entrada para la alimentación de la etapa lógica, finalmente X1-3 es la conexión de tierra.

Adicionalmente se dispone de la bornera X5 la cual presenta un voltaje que es proporcional a la corriente en la armadura de motor. La proporcionalidad queda determinada por el valor del resistor R1 que en este caso es de 0.5 ohmios 5W lo que nos da una proporción de 2 amperios por voltio.

d) Análisis de precisión en la generación de la señal de control

La señal de control calculada en RTSJ es idealmente un voltaje analógico, es decir, si se calculase 10.235 Voltios esto implicaría que se debería aplicar al motor un voltaje de exactamente 10.235 Voltios, sin embargo nosotros no disponemos de una fuente de voltaje controlada si no que tenemos una fuente de voltaje fijo de 24 Voltios y un puente H para aplicar PWM, por lo que los 10.235 Voltios se tendría que escalar un ancho de pulso de PWM que represente dicho voltaje.

Si disponemos de un periodo de PWM de 10ms y un ancho mínimo para generar de 50 us entonces la resolución será de 200 de manera que nosotros podríamos generar voltajes DC múltiplos de 24V/200=0.12 por lo que el voltaje de control de 10.235 voltios se traduciría en un PWM de ancho 10.235/0.12 de 85.29*50us que sería 4.264583ms, pero ya que sólo podemos variar el ancho del pulso de 50 us en 50 us este ancho seria de 4.25ms, es decir se aplicaría durante 100ms (periodo de muestreo) una señal PWM de periodo 10 ms con un ancho de pulso de 4.25ms que no estaría aplicando los 10.235 voltios requeridos sino 10.2 V.

Es por esta razón que la señal de control en forma de voltaje analógico (formato double) no es enviada tal cual al FPGA, ya que aunque el procesador MicroBlaze disponga de los métodos para trabajar en una alta resolución como la que brinda el formato double, esta no se podría reproducir en la salida del actuador sino que estaríamos limitados por la precisión del PWM y el tiempo de retardo en la conmutación del puente H, por lo que no enviamos la señal de control u en formato double si no que hacemos el siguiente cálculo.

$$cmd = u\frac{ResoluciónPWM}{V_s} \tag{4.7}$$

Donde:

ResoluciónPwm, que en nuestro caso sería de 1000.

Vs es el voltaje de alimentación para el motor, en nuestro caso será de 24V

De manera que si reemplazamos estos parámetros en la ecuación 4.7 tenemos:

$$cmd = u\frac{1000}{24} = 41.67u \tag{4.8}$$

$$pwm = [\![cmd]\!] \tag{4.9}$$

La ecuación 4.9 muestra la variable PWM de tipo entero que se envía al FPGA, nótese que sería un desperdicio de procesamiento y tiempo enviar directamente la señal de control en formato double puesto que sólo se requiere el ancho del pulso de la señal PWM.

Debe recordarse que en estos cálculos se están despreciando los tiempos de retardo en la conmutación del puente H, por lo que en la práctica se pierde aun más precisión en la generación de la señal de control.

4.2.2. Adquisición de velocidad angular del motor

La adquisición de la velocidad angular de un motor implica un proceso que comienza con el tratamiento de las señales provenientes del encoder hasta el almacenamiento de la velocidad angular (o alguna variable relacionada con la velocidad) en algún registro que pueda ser leído por el procesador.

El primer paso es el acondicionamiento de la señal que resulta es necesario, puesto que la salida del encoder no es compatible con los niveles lógicos que maneja el FPGA.

El segundo paso es el procesamiento de la señal previamente acondicionada, el cual es realizado por un subsistema implementado en el FPGA, que filtrará y procesará las señales del encoder, el cual finaliza con el almacenamiento de la velocidad angular en algún registro.

El tercer paso es el conexionado del circuito, esto implica la conexión entre el encoder del motor, el circuito de acondicionamiento y el FPGA.

Estos pasos son explicados más detalladamente en los apartados a, b y c que subsiguen.

a) Circuito acondicionador

Tal como se ha visto, sea por el efecto no lineal del fototransistor, ruido en la señal de alimentación o el cableado que genera efectos capacitivos, resulta que el tren de pulsos no es perfecto y es necesario su acondicionamiento.

La solución escogida se basa en el uso de inversores con entrada Schmitt, este tipo de inversores elimina el ruido estableciendo umbrales de estado lógico de alto a bajo y de bajo a alto bastantes separados entre sí además de ofrecer en la salida sólo los estados alto o bajo (fuente o tierra) de manera que el FPGA no cometa errores, a diferencia de la salida

de un encoder que es una señal analógica que no nos asegura tenga en su salida sólo alto o bajo (tierra o fuente).

En el mercado electrónico peruano existe diversidad de integrado que pueden ser usados desde 2 entradas a 8 entradas Schmitt y en encapsulados SMD y DIP.

El integrado elegido es el CD40106B cuya hoja de datos [32] muestra que dispone de 6 entradas Schmitt en encapsulado DIP que puede ser alimentado desde 3V hasta 15V.

b) Diseño de IP para procesador MicroBlaze.

Luego del acondicionamiento de la señal del encoder se dispone de una señal que puede considerarse digital, esta señal presenta una cantidad de pulsos por intervalo de tiempo, el cual es de interés porque se relaciona con la velocidad angular que se quiere medir.

Con el uso de un microcontrolador la solución es bastante conocida y se resume a usar un timer en modo captura de manera que incremente un contador por cada flanco de subida de la señal del encoder que actúa como una señal de reloj, una interrupción es activada cada cierto tiempo (intervalo de muestreo) y la señal del contador es procesada.

En un FPGA aunque se dispone de bloques que permiten dar el rango de señal de reloj a algunas señales externas, sucede que estos son escasos y en general es mejor reservarlos para fines más complejos como la comunicación SPI, Ethernet o manejo de RAM, de manera que la solución abordada no usa estos bloques.

La solución planteada usa un esquema de sobremuestreo, es decir se está muestreando la señal de entrada a una frecuencia de 50MHz de manera que una FSM decida cuando hay un flanco de subida válido.

La implementación está constituida por 1 bloque denominado base de tiempo, 1 par de bloques denominados debouncing, 1 par de bloques denominados Detector de flancos, un bloque denominado contador de flancos, un bloque denominado Giro y concatenación de datos y un último bloque denominado Comunicación con IPIF.

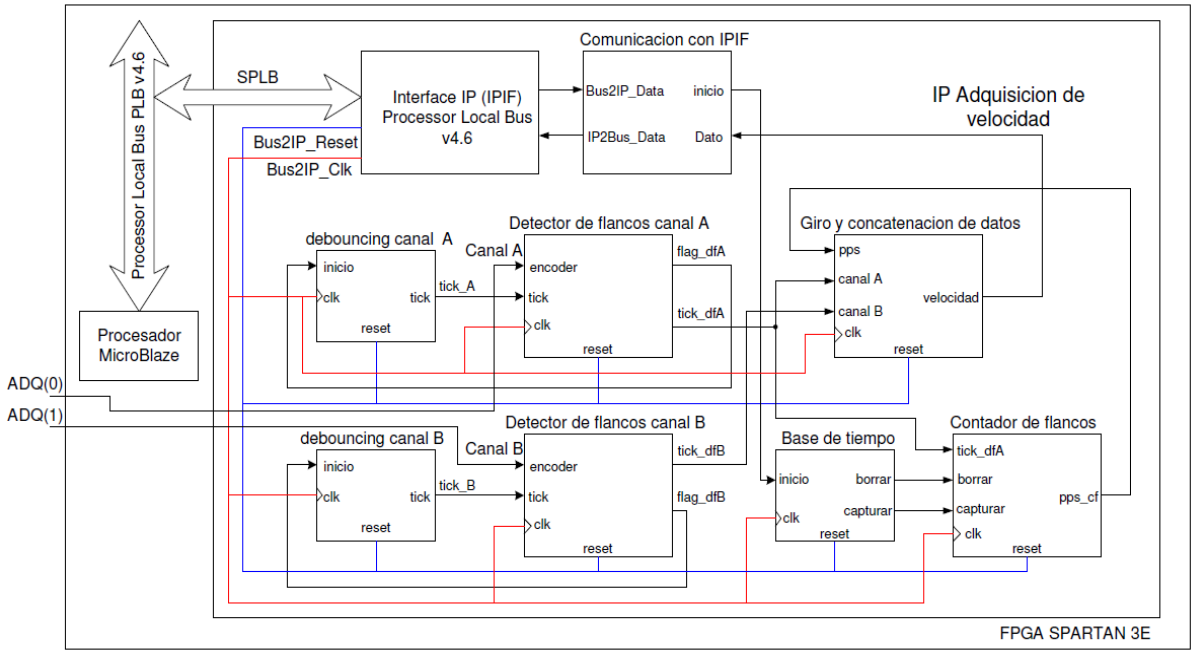


Fig.4.17 Diagrama de bloques de la IP de adquisición de velocidad angular (Fuente: Elaboración propia)

Bloque debouncing.

Este bloque tiene como finalidad apoyar al bloque Detector de flancos a rechazar todos los flancos de subida que tengan una frecuencia mayor a la esperada.

La implementación de este bloque hace uso de una FSM de tres estados el cual puede ser descrito como una base de tiempo que se activa sólo cada vez que su entrada denominada inicio presenta un pulso activo alto, esta base de tiempo tiene una salida denominada tick la cual presenta un pulso de reloj activo alto cada vez que el contador llega a un valor determinado. La carta algorítmica de la FSM de este bloque se muestra en la figura 4.18.

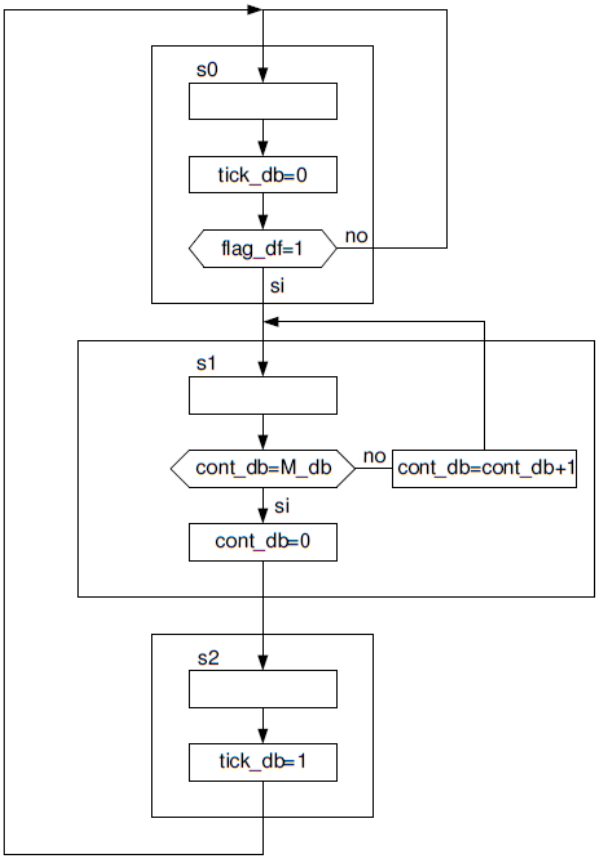


Fig.4.18 FSM para el bloque debouncing (Fuente: Elaboración propia)

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

Inicio, señal que cuando es activada da inicio a la cuenta en este bloque

Salidas:

Tick, señal que presenta un pulso activo alto cada vez que el registro llega a un valor determinado en tiempo de compilación.

Bloque Detector de flancos

Este bloque tiene en su salida flancos de subida validos que servirán para la determinación del sentido de giro y la cuenta de pulsos. La salida flag_df tiene como destino el bloque debouncing, la salida tick_df tiene como destino el bloque Contador de flancos y Bloque giro y concatenación de datos.

La utilidad de los bloques debouncing y Detector de flancos se ve reflejada en la figura 4.19, en esta figura se muestra la señal de un encoder en un ambiente sometido a

ruido y la señal de este mismo encoder a la salida del bloque Detector de flancos. Nótese que sin estos bloques disminuiría la precisión en la adquisición de la velocidad dado que se contaría pulsos no válidos.

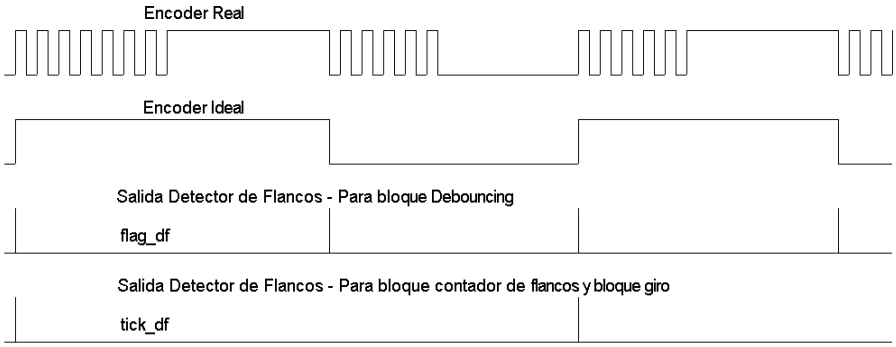


Fig.4.19 Filtrado de ruido usando VHDL (Fuente: Elaboración propia)

La implementación de este bloque hace uso de una FSM de 6 estados la cual tiene como entradas la señal del encoder y la señal del bloque debouncing denominada tick_db.

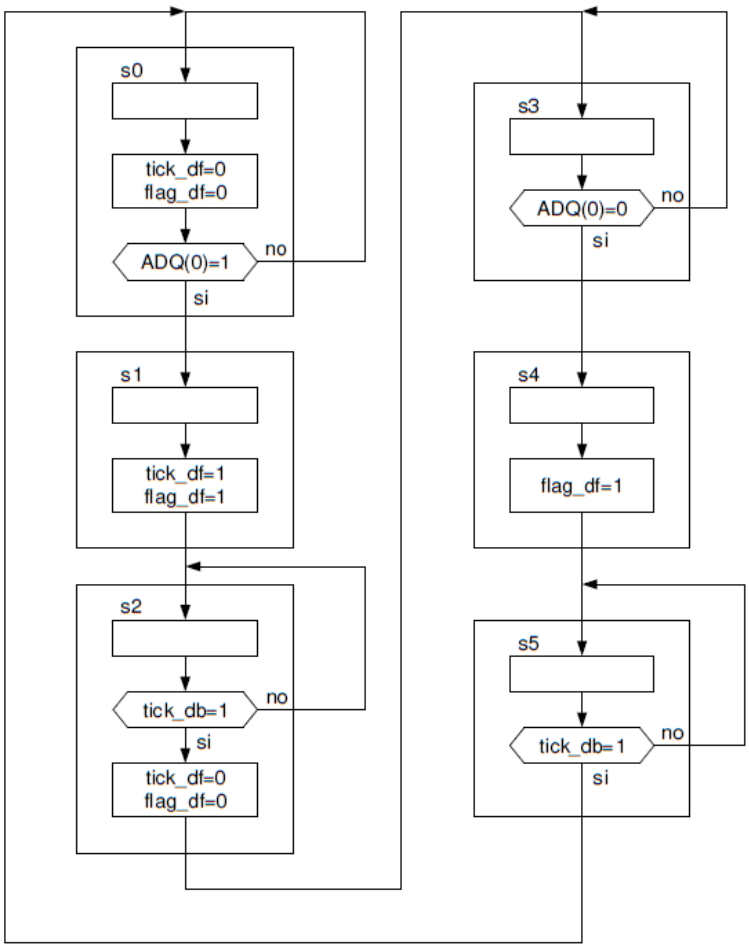


Fig.4.20 FSM para la detección de flancos (Fuente: Elaboración propia)

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

Encoder, señal que contiene una salida del encoder

Tick, señal que en algunos estados indica cambio de estado.

Salidas:

Tick_df, señal que contiene flancos de subida validos obtenidos a partir de la señal del encoder.

Flag_df, señal que contiene flancos de subida validos que indican el inicio de la cuenta en el bloque debouncing.

Bloque base de tiempo

Este bloque tal como lo indica su nombre es una base de tiempo con entrada de activación, es decir empieza su cuenta sólo cuando su entrada es activada con un pulso de reloj activo alto.

Estos bloques base de tiempo sirven para establecer el tamaño de la ventana de tiempo sobre la cual se contaran los pulsos.

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

inicio, señal que cuando presenta un pulso activo alto inicia la cuenta en este bloque

Salidas:

borrar, esta señal presenta un pulso de reloj activo alto cada vez que se inicia un intervalo de muestreo.

capturar, esta señal presenta un pulso de reloj activo alto cada vez que se finaliza un intervalo de muestreo.

Bloque contador de flancos

Este es un circuito secuencial regular está compuesto por dos registros de desplazamiento de 16 bits y un sumador de igual tamaño.

Un registro tiene como finalidad almacenar el valor del contador el cual ira incrementándose según los flancos de subida brindados por el bloque detector de flancos.

El otro registro sirve para retener el valor del contador durante todo un intervalo de muestreo.

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

inicio, señal que cuando presenta un pulso activo alto inicia la cuenta en este bloque

Salidas:

Tick, señal presenta un pulso de reloj activo alto cada vez que el registro de este bloque llega a un valor que debe ser definido en tiempo de compilación.

Bloque Giro y concatenación de datos

Este bloque concentra los datos y señales generados en otros bloques, y tiene como salida un dato de 32 bits que contiene la magnitud y velocidad del eje del motor que se está midiendo.

De los 32 bits que se disponen como ancho del bus, se usará del bit 24 al bit 31 para la definición del sentido y del bit 8 al bit 23 para la determinación de la magnitud.

Siendo el formato de trabajo del procesador hacer que se enumeren los bits de izquierda a derecha comenzando desde cero además de establecer como el bit más significativo el primero de la izquierda tendremos que si la IP core obtiene una lectura de 7 pulsos en la ventana de tiempo estando ADQ(0) (canal A) adelantado respecto de ADQ(1) (canal B) el procesador recibirá 0x00000702:

Bit 0-7	Bit 8-15	Bit 16-23	Bit 24-31
0 1 2 3 4 5 6 7	**8 9 10 11 12 13 14 15**	**16 17 18 19 20 21 22 23**	**24 25 26 27 28 29 30 31**
No en uso	**Magnitud**	**Magnitud**	**Sentido**
0000 0000	**0000 0000**	**0000 0111**	**0000 0010**
0x00	0x00	0x07	0x02

TABLA Nº 4.3 Formato de envío de la velocidad desde la IP core al procesador

Entradas:

Bus2IP_Clk, esta es una señal de reloj de 50 MHz

Bus2IP_Reset, esta es una señal de reset asíncrono

canal_A, canal A del encoder en cuadratura

canal_B, canal B del encoder en cuadratura

tick_dfA, salida del bloque detector de flancos del canal A

tick_dfB, salida del bloque detector de flancos del canal B

pps_cf, es el valor del registro que contiene la cuenta de pulsos en una ventana de tiempo.

Salidas:

tx, dato de 24 bits que contiene la magnitud y sentido de la velocidad del eje del motor

Bloque Comunicación con IPIF

Este bloque se encarga de recibir y procesar los datos enviados por el procesador para inicializar la cuenta en la base de tiempo denominada bt1.

Así también este tiene como finalidad enviar los datos adquiridos en un formato definido en el bloque anterior.

Entradas:

Bus2IP_Data, conjunto de señales que contiene la información que envía el procesador a la IP core.

Dato, dato de 24 bits que contiene la información de la magnitud y sentido de la velocidad del eje del motor.

Salidas:

IP2Bus_Data, conjunto de señales que contiene la información de que se quiere enviar al procesador.

inicio, señal que cuando se encuentra activo alto inicializa esta IP core.

Nota: El código VHDL de esta IP así como un resumen de la síntesis del circuito se encuentra en el anexo F

c) Conexionado de circuito

La conexión del encoder al circuito integrado CD40106B y de este al FPGA se muestra en la figura 4.21. Para la conexión en el FPGA se usa el header J4 que dispone de conexión a 3.3V, tierra y 4 pines del FPGA de los cuales se usa F8 y E8.

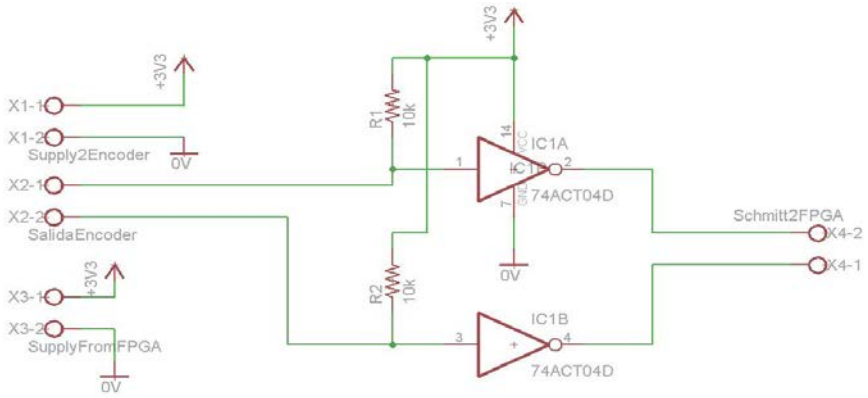


Fig.4.21 Circuito electrónico para el acondicionamiento de las señales en cuadratura provenientes del encoder incremental.

d) Análisis de precisión en la adquisición de la velocidad angular

Para hacer este análisis recordemos la visión global del sistema tal como se puede ver en la figura 2.1 para luego mostrar la secuencia de los procesos de comunicación y de cálculos.

El procesador MicroBlaze solicita a la IP ADQ un número entero denominado pps (pulsos por segundo) que estará relacionado con la velocidad angular conforme a la siguiente ecuación:

$$\omega = pps\,\frac{1}{ppr}\,\frac{1}{\Delta t(s)} \tag{4.10}$$

Donde:

pps es el número entero procesado por la IP ADQ

ppr es el número de orificios en el disco del encoder

Δt es la ventana de tiempo dentro de la cual se cuentan los pulsos del encoder

Reemplazando entonces los parámetros ppr que según especificación del motor es de 400 y Δt que usaremos 0.1s obtenemos:

$$\omega = pps\,\frac{1}{400}\,\frac{1}{0.1}\,\frac{60}{1min} = 1.5pps \tag{4.11}$$

De acuerdo con el análisis hecho es imposible obtener una lectura de 1000 RPM, sino que se leerá 1000.5 RPM, esto es, la medida de velocidad tiene una precisión de 1.5 RPM por lo que la velocidad sólo puede ser medido en valores discretos de 0 RPM, 1.5 RPM, 3.0 RPM y así sucesivamente, de manera que existe un límite en la precisión de la variable.

Este número entero pps es el que el FPGA envía a la PC remota, esta última obtiene la velocidad de acuerdo con la ecuación 4.11 en formato double, sin embargo debe notarse

que aunque el proceso RTSJ trabajara con la velocidad en formato double no implica una mejor precisión si no que debe recordarse como esta variable fue medida.

El proceso RTSJ hará uso de la velocidad y la señal de control para estimar las variables de estado, luego calculará la señal a la salida del integrador y finalmente calculará la señal de control.

4.3. Gráfico de datos en ventana Java

La implementación de Java en tiempo real no permite el uso de objetos dibujables en pantalla por lo que este programa que se denominará GraficoJAVA no hará uso de RTSJ.

El programa GraficoJAVA graficará en pantalla los datos que envíe el programa *TelecontrolRTSJ*, este ultimo enviará la velocidad en RPM por lo que *GraficoJAVA* deberá hacer el escalamiento apropiado para mostrar el dato.

En tiempo de compilación se le asignará a este programa un puerto UDP por el cual recibirá los datos, este puerto es el 4407 y se debe asegurar que el programa *TelecontrolRTSJ* envíe los datos a este puerto.

4.3.1. Implementación de Builder.java y RecepcionUDP.java

Esta clase es la principal en cuanto a jerarquía ya que es la encargada de iniciar un objeto de la clase RecepcionUDP.java el cual es un proceso de tipo Thread que será disparado por eventos periódicos.

La clase RecepcionUDP.java genera un proceso que luego de inicializar algunos parámetros entra en un bucle infinito que espera por un dato por un puerto UDP y luego hace el llamado a otros objetos para graficar el dato recibido.

El proceso de recepción de tramas UDP y la obtención del dato de la trama es idéntico al realizado en la clase ComunicaciónUDP.java del subcapítulo 4.1.2.

En esta clase se hace un escalamiento para graficar datos de manera que el eje “y” este en unidades de RPM, siendo 3000 RPM y -3000 RPM los máximos valores que bajo este escalamiento puede graficarse.

4.3.2. Implementación de VentanaCerrable.java y PanelDibujo.java

Las clases VentanaCerrable.java y PanelDibujo.java están basadas en las clases de igual nombre que se pueden encontrar en [33].

El método update es redefinido para graficar en el panel creado, para disminuir el parpadeo se hace de dos buffer, uno para la grafica presentada y otra para la grafica a

presentarse, de manera que cuando la grafica a presentarse se encuentre lista esta se haga visible [34].

Así también en este método se dibuja líneas referenciales en el eje de las ordenadas y abscisas para una mejor lectura de los datos en el grafico.

Este método hace uso del método del doble buffer para disminuir el efecto de parpadeo, esto es, primero se dibuja en un objeto todos los elementos que contendrá, una vez finalizado el dibujo este se hace visible.

4.3.3. Implementación de Dibujable.java, Implementación de Linea.java y LineaGrafico.java

La interface Dibujable.java [33], la cual es un conjunto de declaraciones de métodos sin definición que nos será útil para la implementación de la clase LineaDibujable.java.

La clase Linea.java [33] define la forma que debe tener un objeto de tipo línea, esta clase implementa un constructor que requiere 4 parámetros, esto es, los pares ordenados de los extremos de la línea.

La clase LineaGrafico.java [33] implementa la interface Dibujable.java y hereda de de la clase Linea.java, dándoles los métodos necesarios para ser dibujados.

Finalmente la ejecución de este programa en conjunto con la ejecución de los programas *TelecontrolRTSJ* y el programa en el FPGA nos da el resultado mostrado en la figura 4.22.

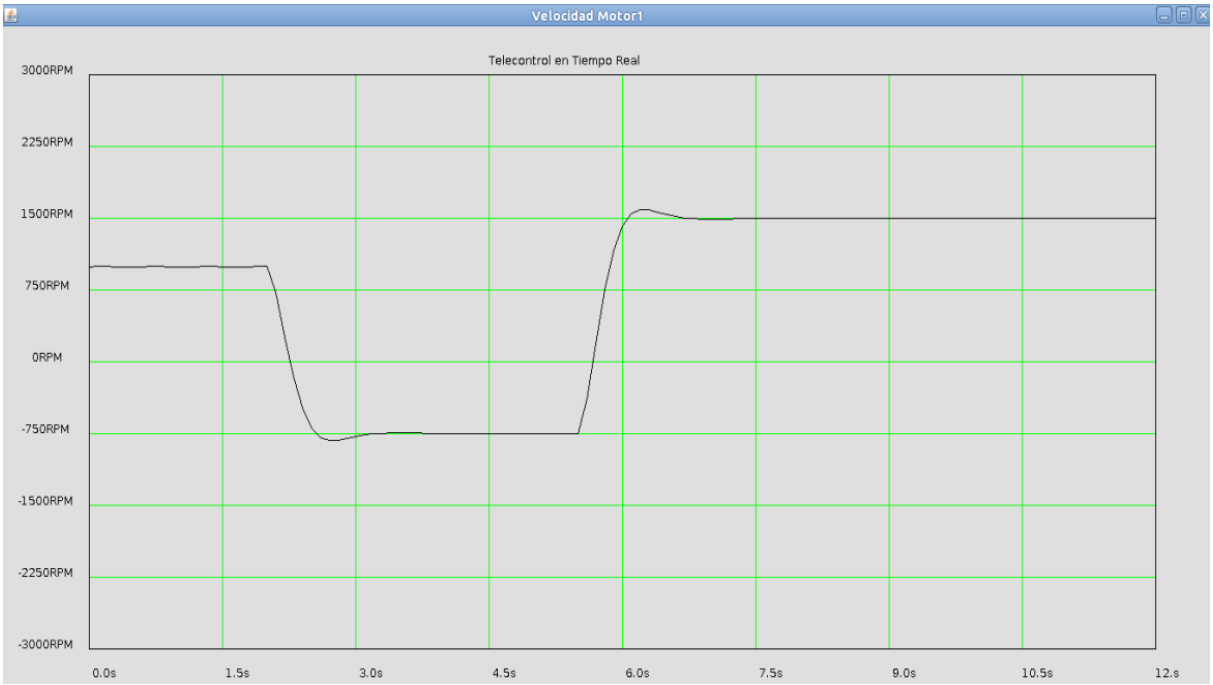


Fig.4.22 Resultados del programa GraficoJAVA

CAPÍTULO V
PRUEBAS Y VERIFICACIÓN DEL SISTEMA

5.1. Obtención del modelo de un motor DC

El diseño de un controlador comienza con el conocimiento de la planta que se desea controlar, por lo que antes de diseñar el controlador es necesario saber o estimar la función de transferencia de la planta.

5.1.1. Respuesta al escalón de un sistema LTI

Para la obtención del modelo del sistema a controlar (motor DC) haremos uso del concepto de "respuesta al escalón de un sistema LTI", el cual consiste en proporcionar al sistema una entrada escalón y estudiar su salida durante un tiempo adecuado (generalmente hasta que la salida se estabilice).

Conociendo la naturaleza del sistema podemos determinar cualitativamente sus parámetros tales como número de polos y ceros así como un posible tiempo de retraso para obtener un sistema como el descrito en la ecuación 5.1.

$$G_{(s)} = k\frac{(s-z_0)\cdot(s-z_1)\cdots(s-z_{m-1})\cdot(s-z_m)}{(s-p_0)\cdot(s-p_1)\cdots(s-p_{n-1})\cdot(s-p_n)}e^{-T_d s} \qquad (5.1)$$

Con el sistema determinado cualitativamente se debe hallar un conjunto de parámetros (polos, ceros y tiempo de retraso) que al ser afectados por una entrada escalón nos dé la salida previamente registrada.

Nótese que dependiendo de la naturaleza del sistema este proceso puede ser bastante complejo de manera que un algoritmo en un computador sería de mucha ayuda en la identificación de estos parámetros.

5.1.2. Captura de de datos

Para obtener el modelo de un motor DC, se ha escogido el método del análisis de la respuesta del sistema ante una o varias entradas escalón. Para esto propósito, debe modificarse el método *handleAsyncEvent* de la clase *Controlador* en el programa

TelecontrolRTSJ y seleccionar el método *Modelamiento* tal como se muestra en la figura 5.1.

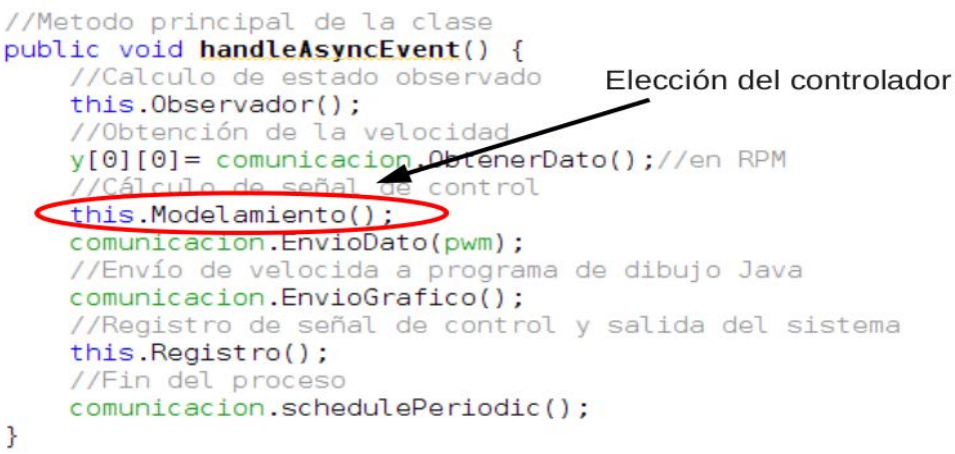


Fig.5.1 Elección del método Modelamiento.

Los datos obtenidos son almacenados en un archivo con extensión .dat el cual puede ser encontrado en la carpeta donde se encuentran las clases de este programa.

Para su exportación a MATLAB debe tenerse en cuenta que la primera columna representa la señal de control y la segunda columna representa la salida de la planta.

5.1.3. Identificación de parámetros usando *Identification Toolbox* de MATLAB

El uso de Identification Toolbox de MATLAB para identificar los parámetros de una planta requiere una aproximación cualitativa de la función de transferencia de la planta y la respuesta de la planta ante una señal de excitación (escalón, impulso, rampa, etc.).

Para iniciar una sesión en el Identification Toolbox (en adelante Ident) se escribe en la línea de comando de MATLAB lo siguiente [26].

```
>> ident
Opening System Identification Tool ..... done.
```

Luego de iniciar la sesión en el Ident se debe ingresar la señal de excitación y la señal de salida, estos deben ser vectores de igual tamaño y deben estar definidos en el WorkSpace de MATLAB.

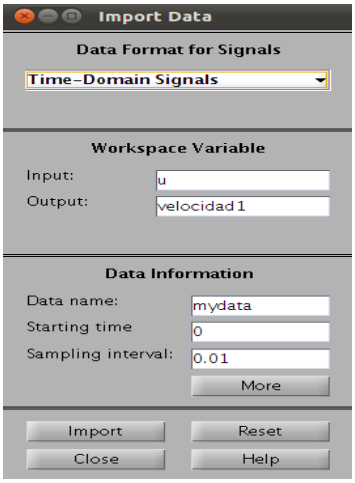


Fig.5.2 Ingreso de vector de excitación, vector de salida, tiempo de inicio y tiempo de muestreo

Una vez importados los datos seleccionamos la pestaña “Estimate” y seleccionamos “Process Model”

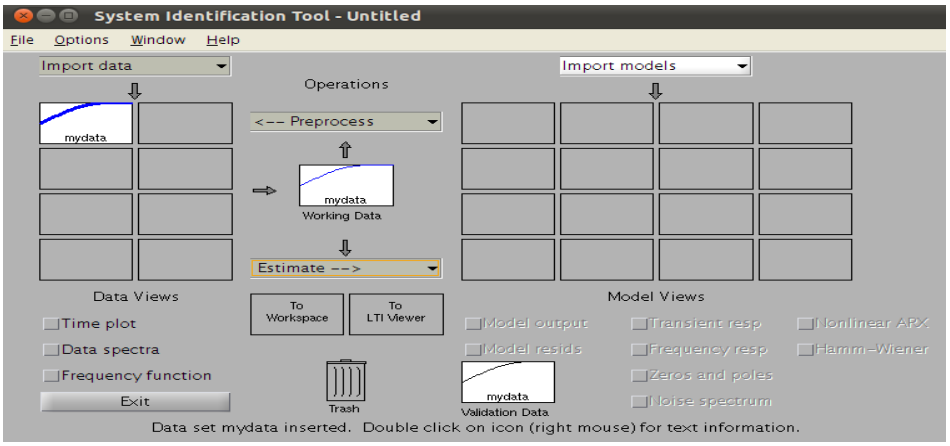


Fig.5.3 Elección del tipo de sistema a modelar

En la ventana de “Process Model” debemos ingresar el número de polos, numero de ceros, naturaleza de los polos y habilitar o no la introducción de un retraso.

En nuestro caso sabemos que la función de trasferencia de la velocidad del eje de nuestro motor con respecto a una entrada de voltaje es una función de transferencia de orden 2, así también debemos indicar que el sistema se encontraba en reposo (vector de espacio de estados en estado cero) y por último pulsamos en el botón “Estimate” quedando nuestra ventana “Process Model” como en la siguiente figura.

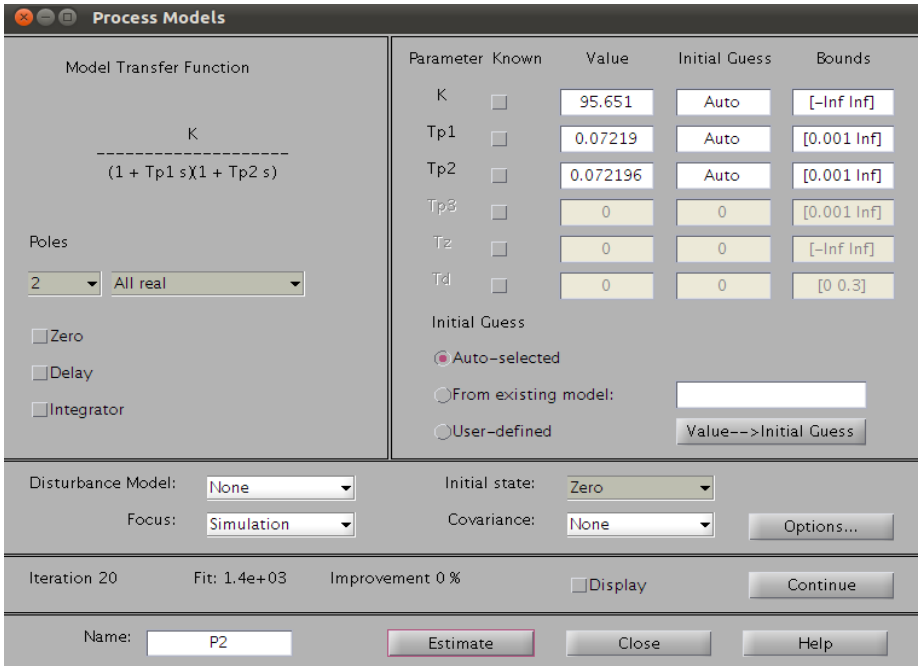


Fig.5.4 Identificación de parámetros para un sistema de segundo orden sin tiempo de retardo y estado inicial nulo

5.1.4. Verificación de la función obtenida

El siguiente paso es dar una forma estándar a nuestra función de transferencia y luego comprobar que las respuestas al escalón de nuestro motor real y de nuestra función de transferencia hallada se aproximen. Los parámetros hallados por Ident son:

```
Process model with transfer function
                Kp
G(s) = ------------------
        (1+Tp1*s)(1+Tp2*s)


with   Kp = 95.651
      Tp1 = 0.07219
      Tp2 = 0.072196

Estimated using PEM using SearchMethod = Auto from data set z
Loss function 1397.49 and FPE 1479.7
Created:       24-Mar-2011 21:00:24
Last modified: 24-Mar-2011 21:00:30
```

Fig.5.5 Salida de MATLAB – parámetros estimados

Y la función de transferencia estimada es:

```
>> num=[95.651];
>> den=conv([0.072196 1],[0.07219 1])

den =

   0.005211829240000   0.144386000000000   1.000000000000000

>> num=num/0.005211829;
>> den=den/0.005211829;
>> Gp=tf(num,den)

Transfer function:
      1.835e04
--------------------
s^2 + 27.7 s + 191.9

>> step(30*Gp)
```

Fig.5.6 Salida de MATLAB – Función de transferencia estimada

Y finalmente obtenemos las respuestas escalón del motor real (línea roja) y de la función de transferencia aproximada (línea azul) en la figura

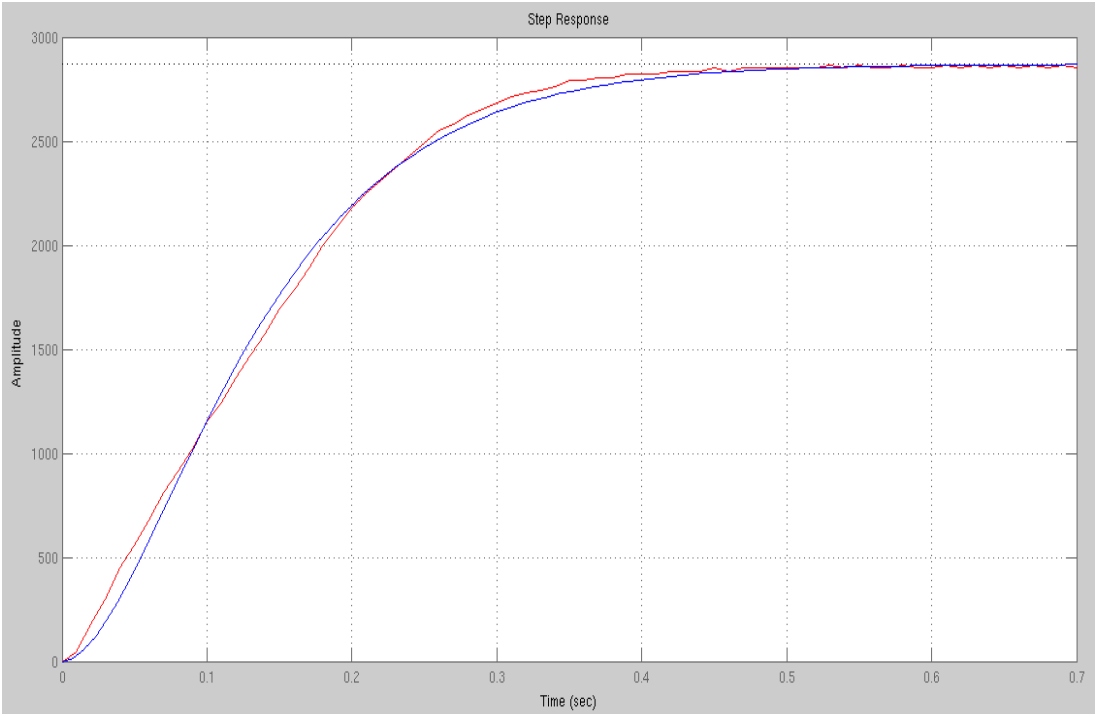


Fig.5.7 Gráfico MATLAB – Salida real y salida estimada

Siendo estas respuestas bastantes aproximadas, se concluye que la identificación de parámetros es satisfactoria y por lo tanto, podemos aceptar como válida la función de transferencia de velocidad del eje del motor en RPM respecto a un voltaje en Voltios.

$$G_{p(s)} = \frac{18350}{s^2+27.703517s+191.871213}.RPM/V \qquad (5.2)$$

Según esta función de transferencia, para una entrada escalón de 1 voltio el sistema se estabiliza en aproximadamente 95.63 RPM, de manera que puede resultar útil la siguiente regla de tres simples en los análisis posteriores:

$$\begin{matrix} 1V & \leftrightarrow & 95.5RPM \\ u & \leftrightarrow & \omega \end{matrix} \tag{5.3}$$

5.1.5. Modelo del motor en el espacio de estados

En el espacio de estados el modelo del motor se escribiría como:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} -27.703517 & -191.871213 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 18350 \\ 0 \end{pmatrix} u$$
$$y = (0 \quad 1) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + (0)u \tag{5.4}$$

Y por tanto las matrices A, B, C y D son como sigue:

$$\begin{aligned} A &= \begin{pmatrix} -27.703517 & -191.871213 \\ 1 & 0 \end{pmatrix} \\ B &= \begin{pmatrix} 18350 \\ 0 \end{pmatrix} \\ C &= (0 \quad 1) \\ D &= (0) \end{aligned} \tag{5.5}$$

Puede verificarse que el sistema descrito por las ecuaciones 5.5 es de naturaleza controlable y observable.

5.2. Control de un motor DC en tiempo real basado en sistema con entrada de referencia

5.2.1. Diseño de un controlador en el espacio de estados

Ahora que tenemos el modelo del motor podemos proceder al diseño de un controlador para lo cual seguiremos los siguientes pasos.

a) Transformación de tiempo continúo a tiempo discreto.

Este paso es bastante importante en lo que a conceptos se refiere, es de vital importancia conocer las suposiciones que se hacen para la transformación y así luego poder explicar los resultados que se obtengan en el control.

Consideremos el siguiente sistema en el espacio de estados:

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx + Du \end{aligned} \tag{5.6}$$

El cual se quiere discretizar y cuya forma será la siguiente:

$$\begin{aligned} x_{(k+1)} &= G_{(T)}x_{(k)} + H_{(T)}x_{(k)} \\ y_{(k)} &= Cx_{(k)} + Dx_{(k)} \end{aligned} \tag{5.7}$$

Entonces considerando que la señal de control u se mantiene constante en los intervalos de muestreo [35] (pudiendo cambiar en los instantes de muestreo) se tiene:

$$G_{(T)} = e^{AT}$$
$$H_{(T)} = \left(\int_0^T e^{A\lambda} d\lambda\right) \quad (5.8)$$

De manera que la discretización del sistema queda determinado por la ecuación 5.8, además que las matrices C y D permanecen igual.

Es necesario recalcar que la única y vital condición de este proceso de discretización es que la señal de control "u" sea constante entre dos periodos de muestreo, es decir que la señal de control sea procesada por un retenedor de orden cero antes de ingresar al sistema.

Conociendo los detalles de este proceso se puede hacer uso del Software MATLAB que posee el comando "c2dm" que transforma un sistema continuo a un sistema discreto siguiendo alguna regla de aproximación, en nuestro caso la regla de aproximación es la del retenedor de orden cero.

Entonces considerando las matrices A, B, C y D de 5.5 tenemos:

```
>> [G,H]=c2dm(A,B,C,D,0.1,'zoh')

G =

  -0.096401728473911   -4.802148587724855
   0.025027978424329    0.596961297513330


H =

   0.025027978424329
   0.002100568894472
```

Fig.5.8 Salida de MATLAB – Matriz G y H

$$G = \begin{pmatrix} -0.09640173 & -4.81204858 \\ 0.02502798 & 0.59696129 \end{pmatrix}$$
$$H = \begin{pmatrix} 0.02502798 \\ 0.00210057 \end{pmatrix} \quad (5.9)$$

b) Determinación de la controlabilidad del sistema

Aunque el sistema hallado en tiempo continuo es de naturaleza controlable y observable puede darse el caso que la transformación a tiempo discreto haga perder la dinámica del sistema, es decir el sistema pierda polos y a su vez su controlabilidad, por lo que debe verificarse la controlabilidad para el sistema en tiempo discreto con periodo de muestro de 0.1 s.

Esta verificación se hace sencilla haciendo uso del Software MATLAB:

```
>> Mc=[H G*H]

Mc =

   0.025027978424329  -0.012499984330319
   0.002100568894472   0.001880358036769

>> rank(Mc)

ans =

     2
```

Fig.5.9 Salida de MATLAB – Matriz de controlabilidad y rango

Y dado que el rango de la matriz de controlabilidad es igual que el orden del sistema concluimos que la transformación del sistema a tiempo discreto no ha hecho se pierda la naturaleza controlable.

c) Especificación de la respuesta transitoria y en régimen permanente

Es bastante conocido que existe una relación inversa entre el tiempo de establecimiento y el máximo sobrepico en la respuesta transitoria de un sistema, esto es, a menor tiempo de establecimiento mayor será el máximo sobrepico y viceversa [36].

Es así que escogemos un máximo sobrepico de 10% y un tiempo de establecimiento menor a 1.2s con una banda de tolerancia del 2%.

Consideremos entonces dos polos complejos de la siguiente forma:

$$p_1 = -\zeta\omega_n + j\omega_n\sqrt{1-\zeta^2}$$
$$p_2 = -\zeta\omega_n - j\omega_n\sqrt{1-\zeta^2} \quad (5.10)$$

Entonces el máximo sobrepico nos da una cota para el factor de amortiguamiento:

$$\zeta > \frac{|\ln 0.10|}{\sqrt{|\ln 0.10|^2+\pi^2}} = 0.5911 \quad (5.11)$$

De manera que escogemos:

$$\zeta = 0.7 \quad (5.12)$$

Así también el tiempo de establecimiento nos da una cota para el producto del factor de amortiguamiento y la frecuencia natural no amortiguada.

$$\frac{4}{\zeta\omega_n} < 1.2\ s \quad (5.13)$$

De manera que escogemos:

$$\omega_n = 5.8\ rad/s \quad (5.14)$$

Quedando los polos definidos como:

$$p_{c1} = -4.06 + j4.142$$
$$p_{c2} = -4.06 - j4.142 \quad (5.15)$$

Nótese que este proceso se ha llevado a cabo en el dominio del tiempo continuo [36] de manera que hay que mapear estos polos al plano z.

$$p_{d1} = -0.609966 + j0.268162$$
$$p_{d2} = -0.609966 - j0.268162 \quad (5.16)$$

d) Determinación de la ganancia para la localización adecuada de los polos

Una vez determinados los polos deseados del sistema vamos a determinar la ganancia de realimentación de estados para el siguiente sistema:

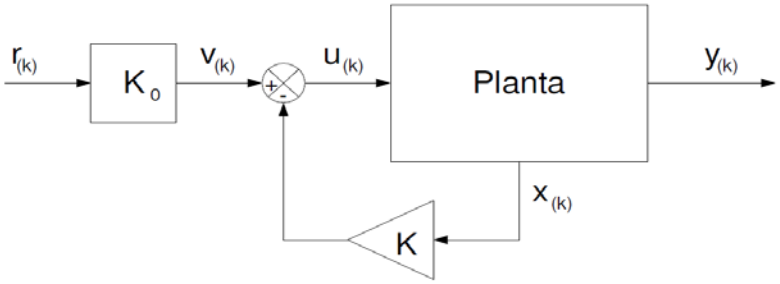


Fig.5.10 Realimentación de estados con entrada de referencia (Fuente: Elaboración propia)

De manera que debemos determinar el vector K y escalar k_0, para esto debemos ejecutar los siguientes comandos en MATLAB:

```
G =

  -0.096401728260098  -4.802148583139818
   0.025027978444791   0.596961298060791


H =

   0.025027978444791
   0.002100568895341

>> Pc=[-4.06+4.142*(1j) -4.06-4.142*(1j)];
>> Pcd=exp(Pc*Tm)

Pcd =

  0.609966033218706 + 0.268161663056478i  0.609966033218706 - 0.268161663056478i

>> Kcd=place(G,H,Pcd)

Kcd =

   1.0e+02 *

  -0.190578408980062  -1.153941039376317
```

Fig.5.11 Salida de MATLAB – Polos deseados y ganancia de realimentación de estados

Con lo cual obtenemos

$$K_{cd} = [-19.05784 \quad -115.3941] \quad (5.17)$$

La ganancia Kr es hallada considerando que para una entrada escalón se debe tener una salida que se establezca en 1.

```
>> [b,a]=ss2tf(G-H*Kcd,H,C,D)
b =
                   0   38.545439229513214   15.210281545623451

a =
   1.000000000000000  -1.219932066437412   0.443969239213780
>> kr=polyval(a,1)/polyval(b,1)
kr =
   0.004167689867159
```

Fig.5.12 Salida de MATLAB – Ganancia Kr

$$K_r = [0.0041676899] \tag{5.18}$$

5.2.2. Diseño de un observador completo de tipo predictivo.

En el apartado anterior encontramos un vector de ganancia para la realimentación de estados, ahora es necesario estimar los estados de manera que el sistema pueda ser controlado.

a) Observabilidad del sistema

Con las matrices G de la ecuación 5.9 y C de la ecuación 5.5 obtenemos la matriz de controlabilidad Mo cuyo rango coincide con el orden del sistema por lo que podemos afirmar que el sistema es observable.

```
>> Mo = [C', G'*C']
Mo =
   1.0e+04 *
         0    0.0459
    1.8350    1.0954
>> no = rank (Mo)
no =
     2
```

Fig.5.13 Salida de MATLAB – Matriz de Observabilidad y rango

b) Elección de los polos del observador

De la teoría de control sabemos que el diseño del observador y el controlador se pueden realizar de manera independiente, de manera que los polos en lazo cerrado que se desea se generen mediante la realimentación de estados se eligen según los requisitos de la respuesta transitoria y de estado estable.

Así también los polos del observador se eligen de manera que la respuesta del observador sea mucho más rápida que la respuesta del sistema.

Dados estos conceptos se eligieran los polos en lazo cerrado en una cantidad tres veces mayor de los polos del sistema mostrados en la ecuación 5.15 con realimentación de estados.

$$p_{oc1} = 3p_{c1} = -12.18 + j12.426$$
$$p_{oc2} = 3p_{c2} = -12.18 - j12.426 \quad (5.19)$$

Y mapeando estos polos al dominio del tiempo discreto con un tiempo de muestreo de 0.1 s tenemos.

$$p_{od1} = -0.09535 + j0.28003$$
$$p_{od2} = -0.09535 - j0.28003 \quad (5.20)$$

c) Determinación de la ganancia del observador

Una vez determinados los polos del observador se procede a determinar la ganancia usando MATLAB.

```
>> Kod=place(G',C',Pod)

Kod =

   1.0e-04 *

  -0.108870609062931   0.16885657991282
```

Fig.5.14 Salida de MATLAB – Ganancia del observador

$$K_r = [-0.000010887 \quad 0.000016886] \quad (5.21)$$

5.2.3. Simulación del sistema

Antes de implementar el controlador y observador en RTSJ es adecuado verificar el desempeño en simulación y así luego comparar los resultados.

a) Simulación usando MATLAB

La simulación es realizada en SIMULINK de manera que son elaborados un diagrama de bloques para la simulación y un script para generar los parámetros que serán usados por SIMULINK.

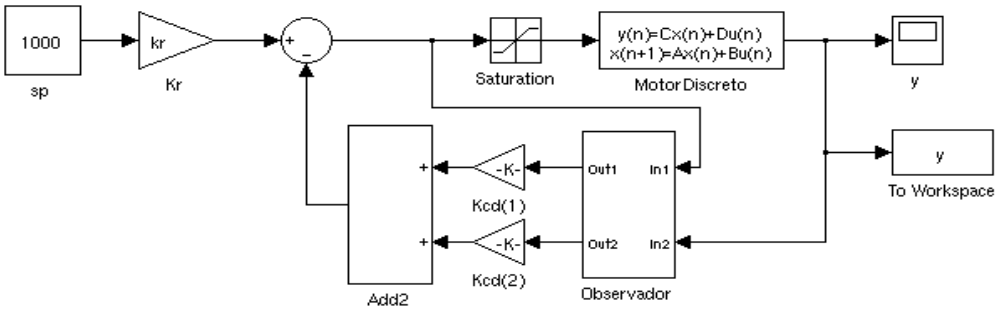


Fig.5.15 Diagrama de bloques del sistema con entrada de referencia

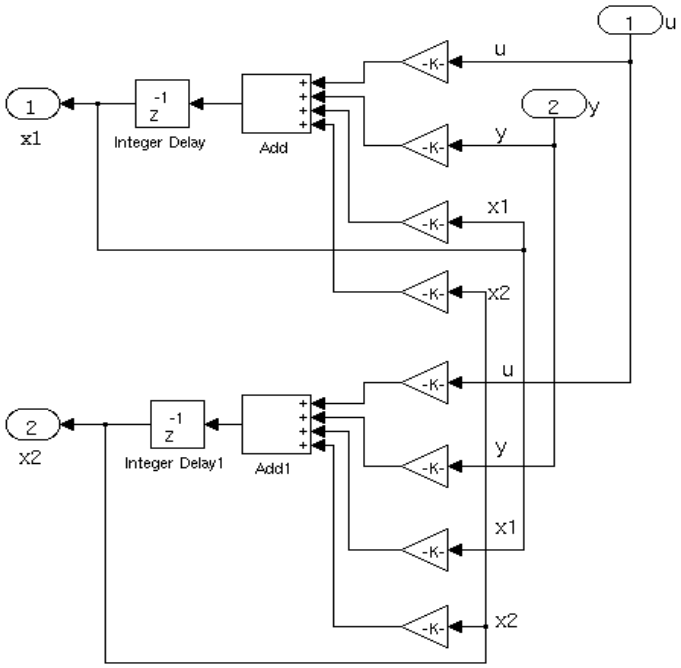


Fig.5.16 Diagrama de bloques del observador

Y tal como se observa en la figura 5.17 se cumple con los requisitos impuestos en la respuesta transitoria ya que el máximo sobrepico es 4.46% y el tiempo de establecimiento es 1.1 s.

.

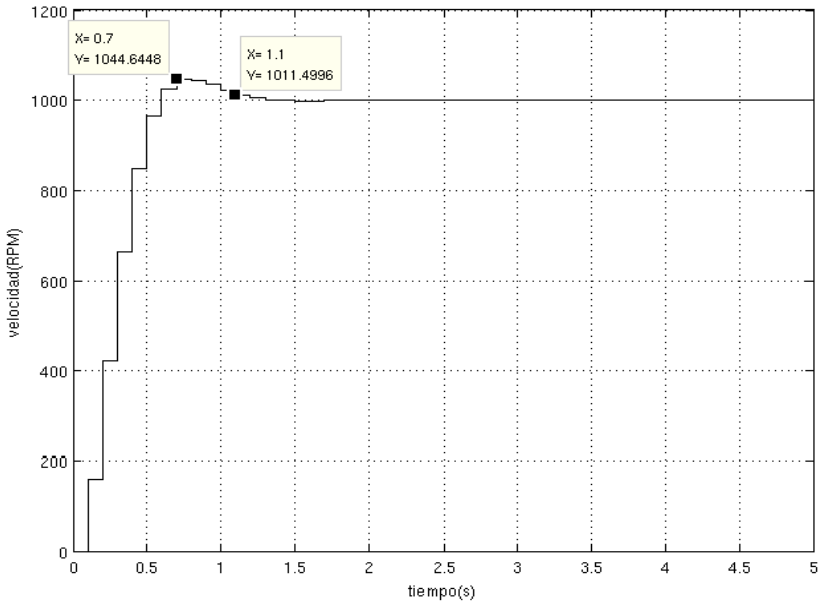


Fig.5.17 Salida del sistema ante un set point de 1000 RPM

Debe recordarse que el software MATLAB opera la salida de la planta, las variables de estado y la señal de control en formato double, sin embargo en el sistema con el motor real, aunque RTSJ permita operaciones en formato double para la implementación del observador y controlador los resultados estarán limitados por la precisión de la medida de la velocidad y la precisión de la señal de control inherentes al sensor y actuador.

b) Necesidad de la sincronización unilateral

Consideremos los programas *TelecontrolRTSJ* y SimulacionFPGA que serán usados sin la implementación de la clase SincronizacionUDP, estos se ejecutan y los resultados se muestran en las figura 5.18 y la figura 5.19, las cuales muestran máximos sobrepico de 4.44% y 8.84% respectivamente.

Dado que las salidas mostradas son los resultados de las ejecuciones de dos programas iguales (luego de quitar SincronizacionUDP no se altera ni *TelecontrolRTSJ* ni SimulacionFPGA), no hay razón aparente para que las respuestas transitorias varíen según cual programa sea lanzado primero, así que observando la figura 5.19 se repara en que en el segundo periodo de muestreo el controlador (programa *TelecontrolRTSJ*) no recibe el dato del sensor (velocidad angular) lo cual se traduce en un excesivo máximo sobrepico de 8.94% que difiere en demasía de lo esperado según simulación en MATLAB.

Un análisis de la figura 5.19 muestra en la secuencia de cálculos que en el segundo periodo de muestreo el programa *SimulacionFPGA* debió enviar un número entero con un valor de “85” que representase una velocidad de 159.375, y puesto que no se envió el

controlador asume que el eje del motor no se movió por lo que en el siguiente periodo de muestreo aumenta la potencia entregada, sin embargo es lógico pensar que el motor si giró y que no se recibió la velocidad porque el paquete UDP no llegó a tiempo.

El aumento en la potencia en la señal de control hace que el motor en el segundo periodo de muestreo gire más rápido de lo previsto y finalmente esto resulta en un máximo sobrepico mayor a lo esperado.

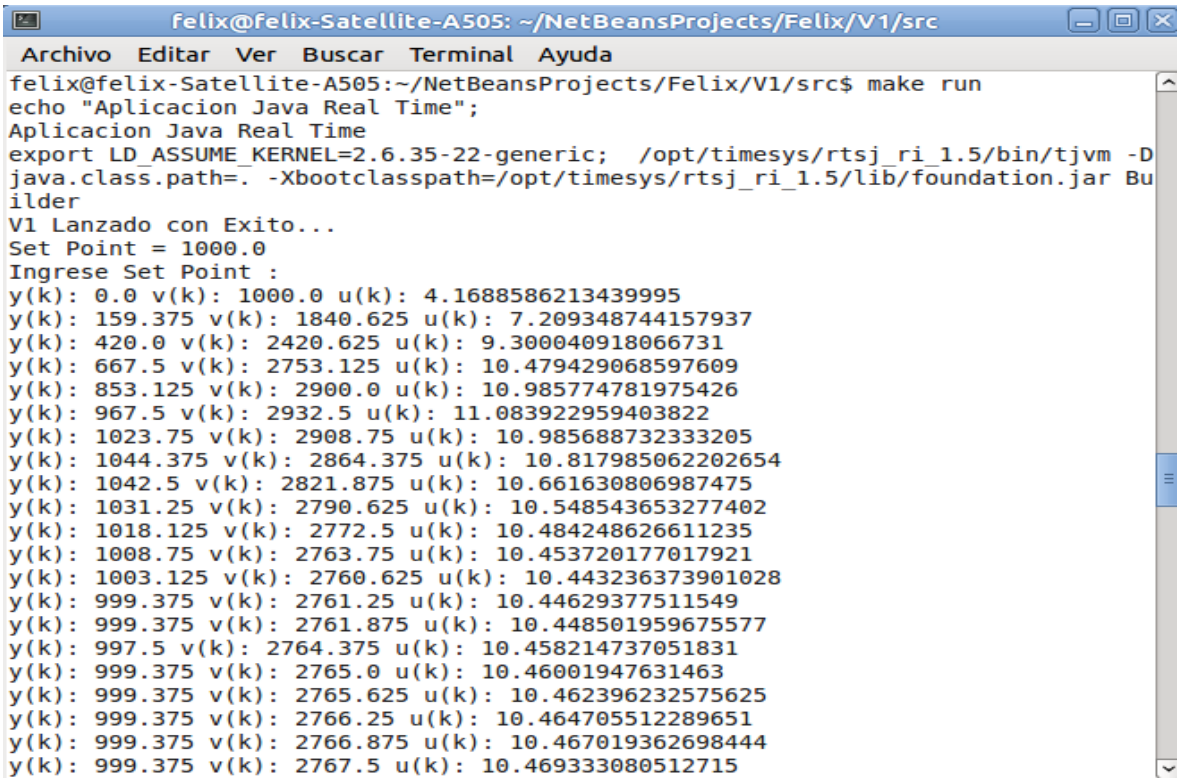


Fig.5.18 Salida eventual del programa TelecontrolRTSJ cuando se lanza primero TelecontrolRTSJ

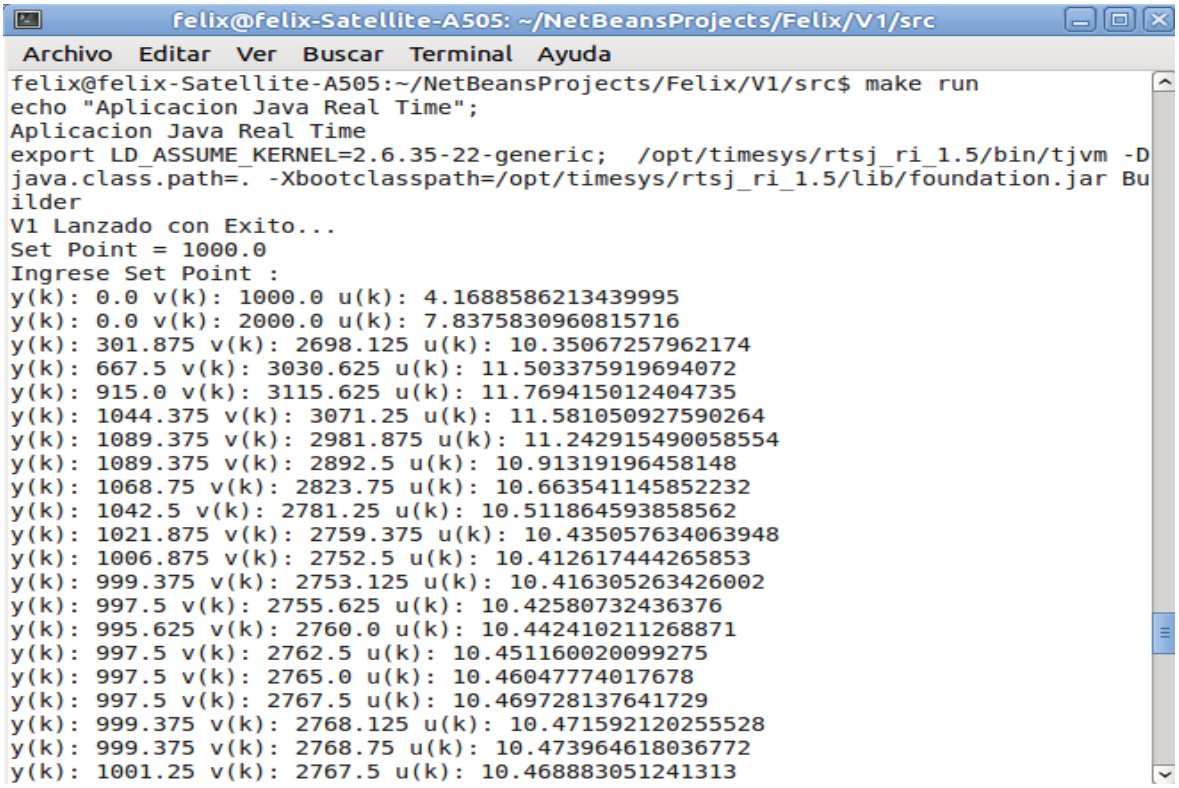


Fig.5.19 Salida eventual del programa TelecontrolRTSJ cuando se lanza primero TelecontrolRTSJ

Dado que la pérdida de este dato no puede deberse a la pérdida de paquetes propias de una red ya que estos programas son ejecutados dentro de un mismo computador (direcciones IP establecidas como localhost), se concluyó que hacía falta una sincronización, es decir, cuando se inicia por ejemplo el programa SimulacionFPGA y luego el programa *TelecontrolRTSJ* se corre el riesgo de que los periodos de inicio se encuentren desfasados y entonces se entreguen y reciban tramas UDP fuera de tiempo.

Puesto que no es recomendable que la precisión de este sistema de control dependa del momento en el cual sean lanzados los programas se decidió implementar la clase SincronizacionUDP, la cual es implementada en el apartado e) de 4.1.2, que dispone de dos métodos denominados ReconocimientoTx y ReconocimientoRx. La implementación de esta clase solucionó los problemas de sincronización tal como se puede observar en todas las siguientes ejecuciones que se realizan en lo que sigue del trabajo tanto con un motor simulado como con un motor real.

c) Simulación usando RTSJ

Haciendo uso del programa *TelecontrolRTSJ* descrito en el subcapítulo 4.1.2 y del programa SimulacionFPGA se puede simular el proceso de comunicación y procesamiento

que se muestra en la figura 2.1, esto es, el programa *SimulacionFPGA* simulará el proceso de adquisición de velocidad, envío de velocidad, recepción de PWM y establecimiento de PWM que realizará el FPGA.

Los programa *TelecontrolRTSJ* y *SimulacionFPGA* son mostrados en los anexos A y B respectivamente, es preciso indicar que se simulará con un periodo de muestreo de 100 ms, una resolución de encoder de 1.5 RPM, una resolución de PWM de 10000 con un periodo de 0.01 ms y una fuente de alimentación de 24 V.

Los resultados se muestran en las figura 5.20, figura 5.21 y figura 5.22, la cuales muestran máximos sobrepicos de 5.75%, 5.80% y 5.75% respectivamente tal como se puede observar en el resumen mostrado en la tabla 5.1.

Set Point (RPM)	Máximo Sobre Pico (%)	Tiempo de establecimiento con el criterio del 2% (s)
1000	5.75	1.1
1500	5.80	1.1
2000	5.75	1.1

TABLA Nº 5.1 Sistema con entrada de referencia implementado con los programas TelecontrolRTSJ y SimulacionFPGA

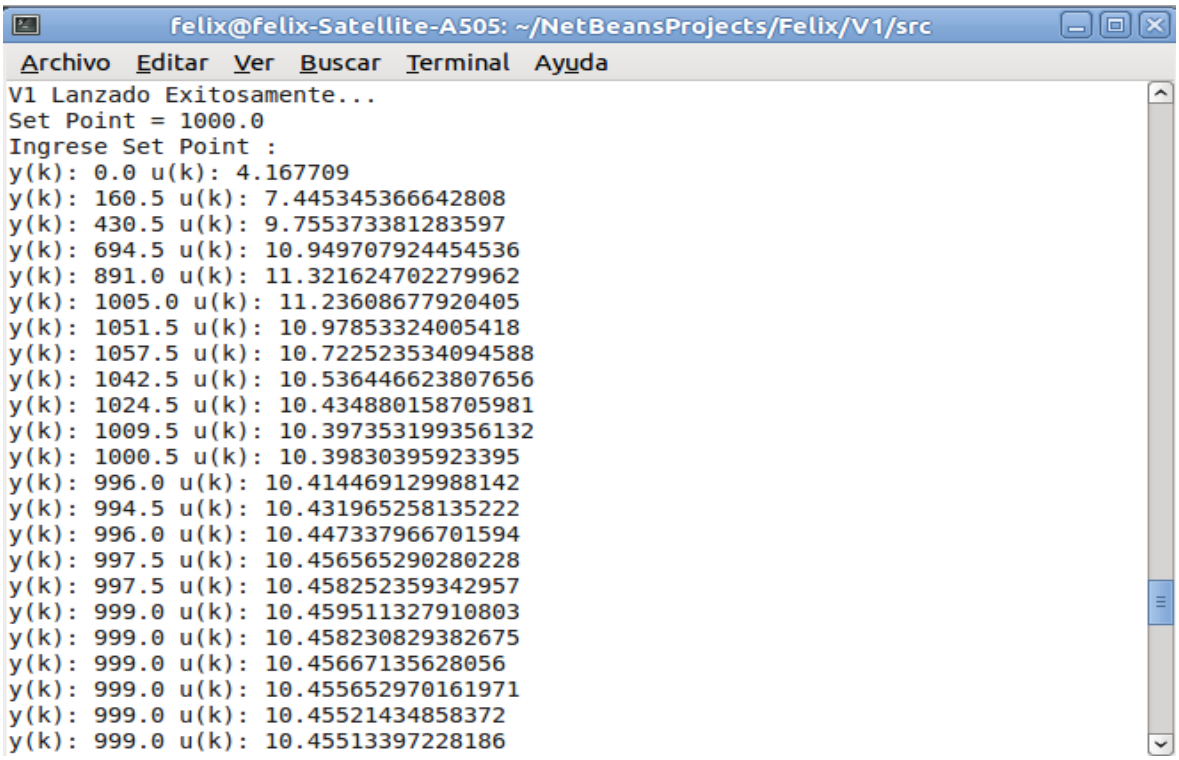


Fig.5.20 Salida del programa TelecontrolRTSJ con un set point de 1000 RPM

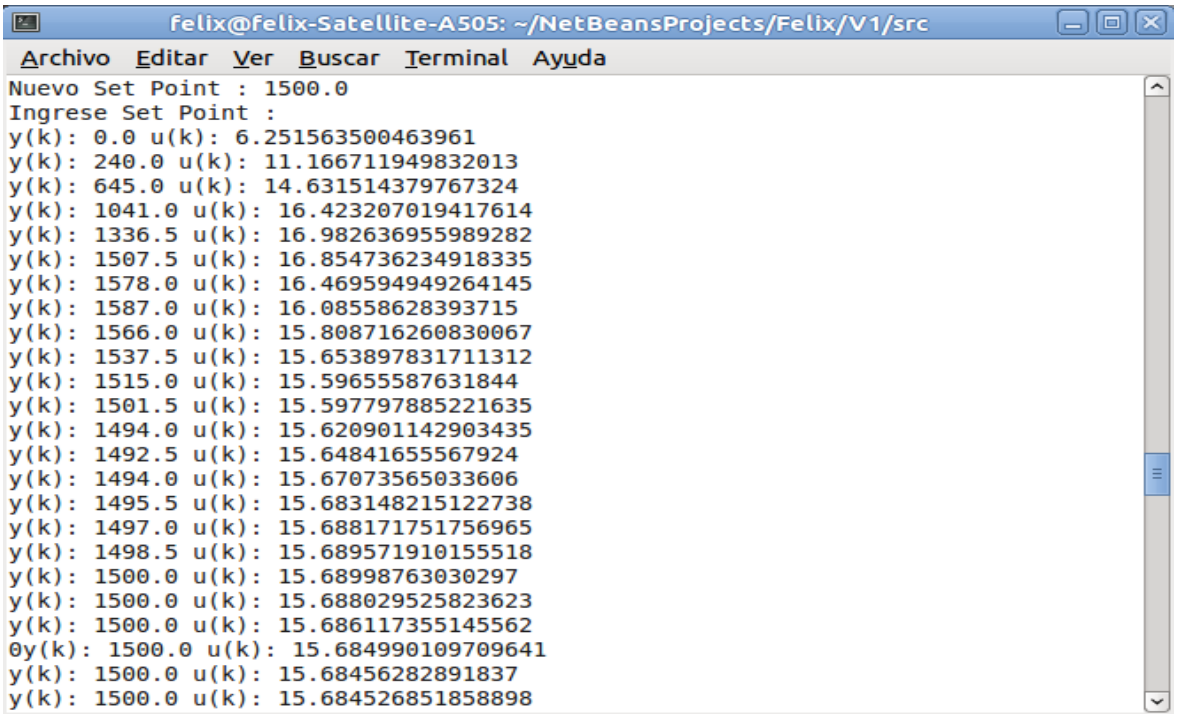


Fig.5.21 Salida del programa Telecontrol RTSJ con un set point de 1500 RPM

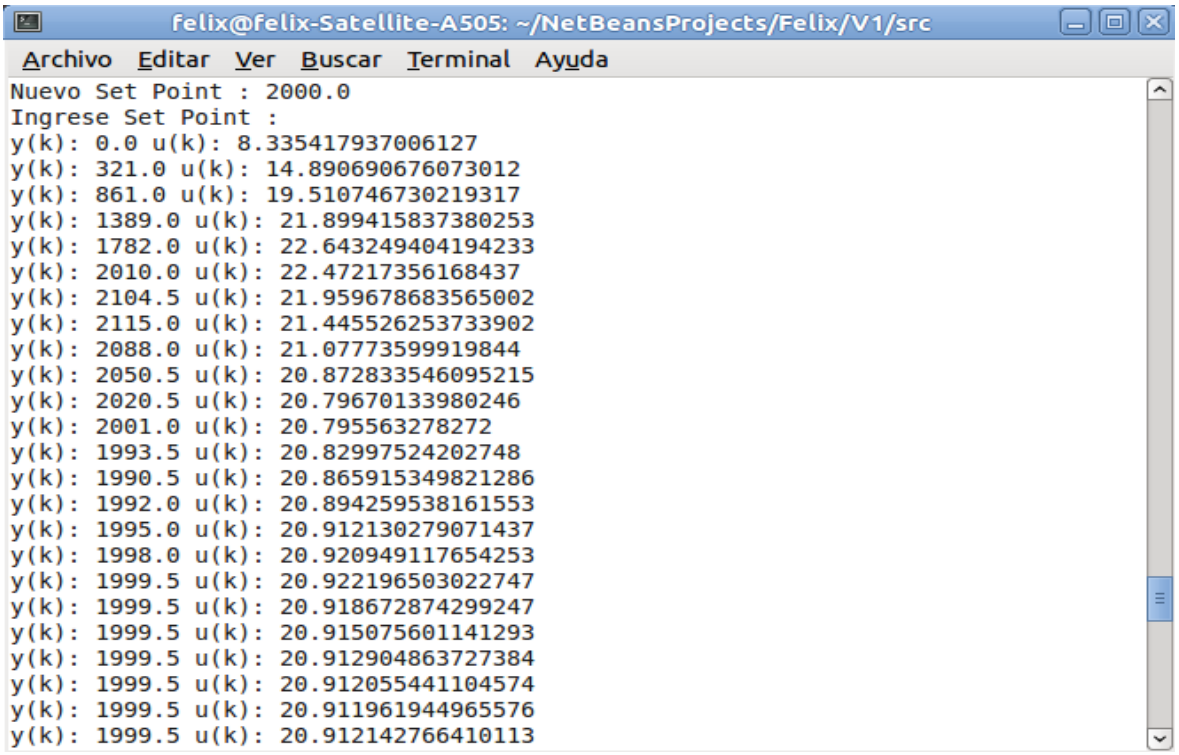


Fig.5.22 Salida del programa TelecontrolRTSJ con un set point de 2000 RPM

Y puesto que los resultados son satisfactorios se puede seguir con la implementación PC – FPGA.

5.2.4. Prueba del sistema de regulación con entrada de referencia con un motor DC real

El programa *TelecontrolRTSJ* que se usará para implementar el controlador y observador es el mismo que el usado en 5.2.3, salvo la modificación de la IP que en vez de establecerla en localhost debe ser establecida con la dirección IP del FPGA la cual es 200.37.46.189 además de elegir el método *Controlador1* tal como se muestra en la figura 5.23.

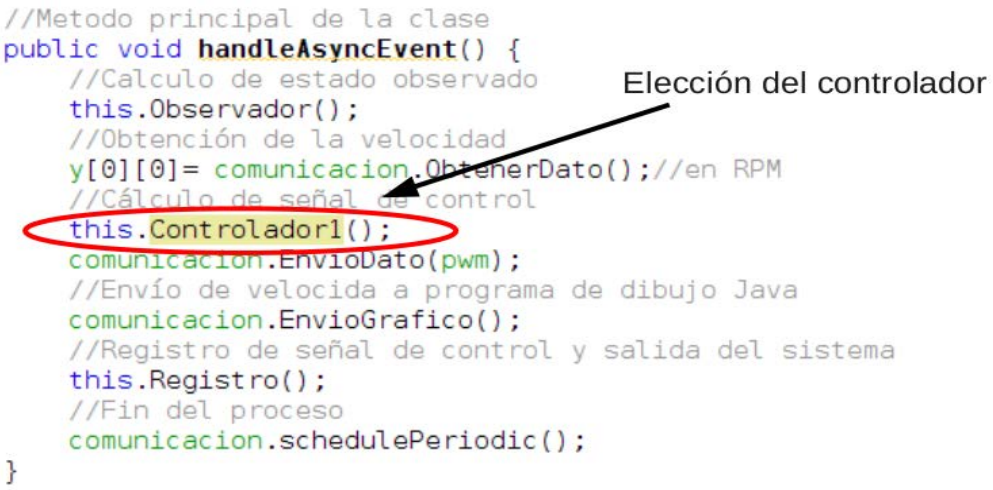


Fig.5.23 Elección del método Controlador1 (sistema de control con entrada de referencia)

El programa en el FPGA usado es el descrito en subcapítulo 4.1.4 cuyo diagrama de flujo y código se muestra se muestra en el anexo D, la fuente de alimentación usada es de tipo **conmutada**, el set point es el establecido desde la PC remota, el voltaje de la fuente de alimentación fue leído con multímetro digital y la corriente con un amperímetro analógico en serie con la fuente.

Los resultados se muestran en las figura 5.24, figura 5.25 y figura 5.26 además la tabla 5.2 muestra un resumen de los datos obtenidos. Nótese que existe un error en estado estable para los tres casos.

Es conocido que para un sistema con realimentación unitaria de tipo 0 existe un error en estado estable, sin embargo en el actual sistema estamos realimentando la salida no para comparación con el set point sino para la estimación de estados de manera que el error en estado estable no puede deberse a la incapacidad del sistema de seguir una entrada escalón, por lo que este error debe atribuirse a otras causas.

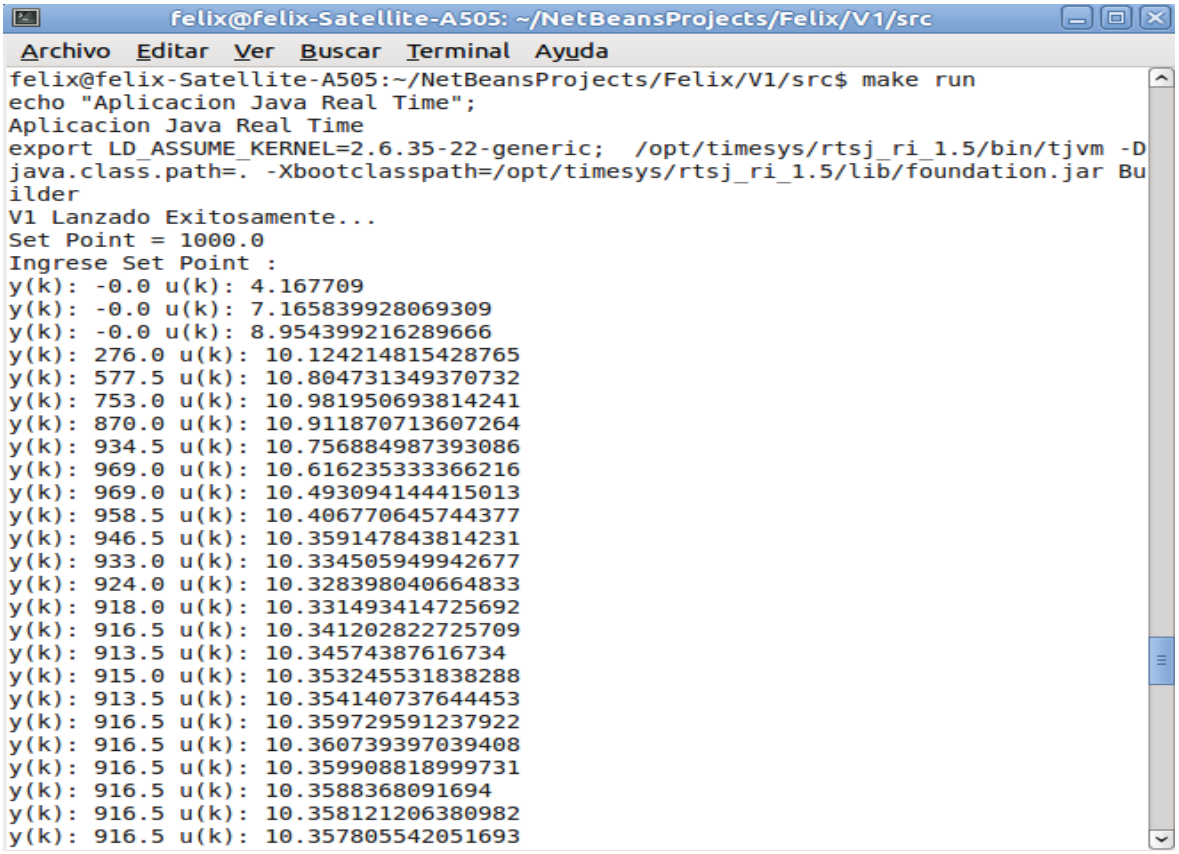


Fig.5.24 Control de velocidad angular del eje del motor con un set point de 1000 RPM usando un sistema de control con entrada de referencia

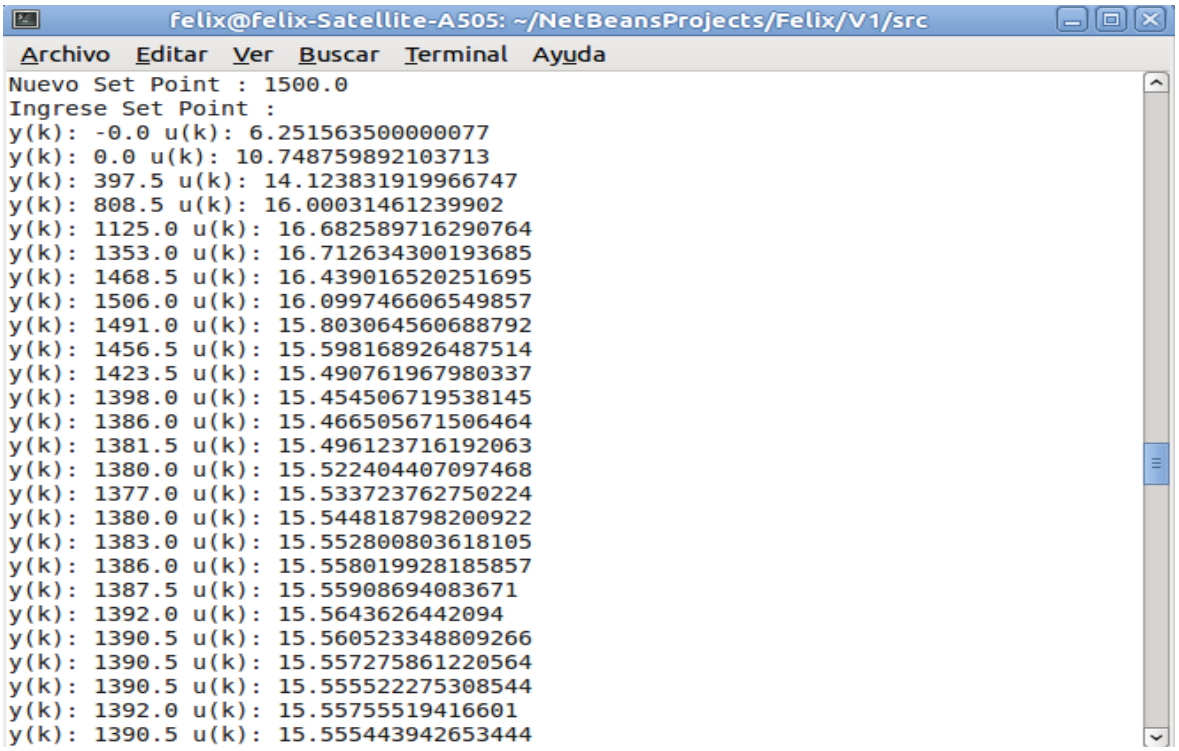


Fig.5.25 Control de velocidad angular del eje del motor con un set point de 1500 RPM usando un sistema de control con entrada de referencia

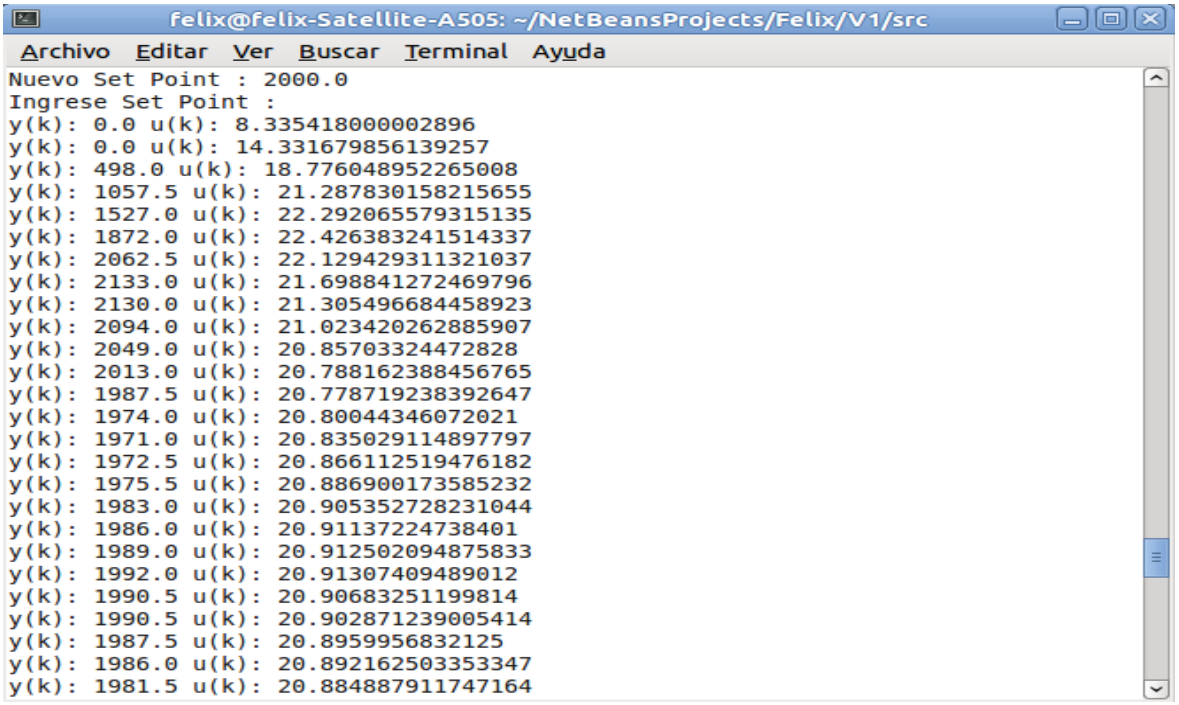


Fig.5.26 Control de la velocidad angular del eje del motor con un set point de 2000 RPM usando un sistema de control con entrada de referencia

Se concluye que el error en estado estable se debe a la caída de tensión en la fuente de alimentación y en los transistores del puente H de manera que al motor no le llega una señal PWM de 24 voltios de amplitud sino una tensión menor que depende de la corriente consumida (efecto de carga y caída de tensión en el puente H).

Set Point (RPM)	Voltaje de la fuente de alimentación (Voltios)	Corriente de la fuente de alimentación (Amperios)	Velocidad Obtenida (RPM)
0	24.10	0.0	0
1000	23.70	0.8	916.5
1500	23.70	0.8	1390.5
2000	24.00	0.4	1986.5

TABLA Nº 5.2 Resumen de resultados al usar un sistema con entrada de referencia

Con los resultados mostrados en la tabla 5.2 se puede hacer otra tabla en la que se añadan el voltaje promedio requerido para obtener la salida deseada, el ciclo de trabajo requerido de la señal PWM para obtener la salida deseada siendo la amplitud de esta señal 24 V, la velocidad obtenida y el voltaje promedio aplicado para obtener la velocidad real.

Nótese que el voltaje requerido para un determinado set point se obtiene de la ecuación 5.3, el ciclo de trabajo requerido de obtiene de dividir el voltaje requerido con la amplitud de la señal PWM que en este caso es 24 V y el voltaje aplicado es el voltaje que aplicado al motor generaría la velocidad real y que es calculado con la ecuación 5.3.

Set Point (RPM)	Voltaje requerido (Voltios)	Ciclo de trabajo requerido (%)	Velocidad obtenida (RPM)	Voltaje real aplicado (Voltios)
1000	10.47	43.63	916.5	9.59
1500	15.70	65.41	1390.5	14.56
2000	20.94	87.25	1986.5	20.80

TABLA Nº 5.3 Cálculo de voltaje promedio real aplicado

Con estos datos es posible calcular la caída de tensión total que se produce desde la salida de la fuente de alimentación hasta los bornes del motor, ya que aunque se debe aplicar 24 voltios en realidad se está aplicando 24-ΔV que multiplicado con el ciclo de trabajo nos da un voltaje promedio que es el que genera la velocidad obtenida.

Set Point (RPM)	Velocidad Real obtenida (RPM)	Voltaje promedio real aplicado (Voltios)	Ciclo de trabajo aplicado (%)	Amplitud de voltaje de PWM (Voltios)	Caída de tensión ΔV (Voltios)
1000	916.5	9.59	43.63	21.98	2.02
1500	1390.5	14.56	65.41	22.26	1.74
2000	1986.5	20.80	87.25	23.83	0.17

TABLA Nº 5.4 Cálculo de la caída de tensión en el puente H

Los cálculos del máximo sobrepico y tiempo de asentamiento pueden hacerse tomando en cuenta la velocidad real en estado estable de la tabla 5.1.

Set Point (RPM)	Máximo Sobre Pico (%)	Tiempo de establecimiento (s)
1000	5.42	1.1
1500	5.80	1.1
2000	5.75	1.1

TABLA Nº 5.5 Respuesta transitoria usando un sistema con entrada de referencia para el control de velocidad de un motor real y para diferentes set point

De la tabla se puede observar que los máximo sobrepico están en el margen de lo esperado al igual que los tiempos de establecimiento.

5.2.5. Análisis de las perturbaciones en un sistema con entrada de referencia.

Son dos los tipos de perturbaciones que serán analizados, el primero corresponde a los efectos de una señal de perturbación en la entrada de la planta y el segundo en la salida de la planta.

a) Perturbación a la entrada de la planta

Este tipo de perturbación es aquel que altera la señal de control que debe ingresar a la planta.

En este trabajo se puede atribuir esta perturbación principalmente al efecto de carga en la fuente de alimentación y caída de voltaje en el puente H los cuales dependen a su vez de la velocidad a la que este girando el motor y la carga que este último tenga.

De la figura 5.27 se puede escribir la siguiente ecuación:

$$x_{(k)} = u_{(k)} + n_{(k)} \tag{5.22}$$

Donde *x* es la señal de control alterada que ingresa a la planta, *u* es la señal de control que debería aplicarse a la planta y *n* es la perturbación.

Dado que es una señal PWM la que ingresa a la planta y que la amplitud máxima de esta señal es disminuida por el efecto de carga se considera que el voltaje promedio aplicado a la planta es:

$$x_{(k)} = (V_s + \Delta V)\delta \tag{5.23}$$

Donde:

Vs es el voltaje que en condiciones ideales seria la amplitud de la señal PWM.

ΔV es la variación en la amplitud del voltaje de la señal PWM y

δ es el ciclo de trabajo de la señal PWM.

Comparando la ecuación 5.22 y la ecuación 5.23 tenemos que:

$$u_{(k)} = V_s \cdot \delta$$
$$n_{(k)} = \Delta V \cdot \delta \qquad (5.24)$$

De donde se observa que la señal de perturbación *n* es directamente proporcional a la variación del voltaje y al ciclo de trabajo de la señal PWM.

Para efectos prácticos de simulación se considerará la variación de voltaje como una señal escalón y el ciclo de trabajo constante y del valor requerido para un set point determinado.

La figura 5.27 muestra el diagrama de bloques a usar para simular los efectos de la perturbación. Para poder comparar los resultados de la simulación con los obtenidos en el 5.2.4 se considerará un set point de 1000 RPM y un voltaje de alimentación de 24 V.

Según la ecuación 5.3 para obtener una velocidad de 1000 RPM se requiere un voltaje de 10.47 V, para esto es necesario una señal PWM que tenga un ciclo de trabajo de 43.63% con una amplitud de voltaje de 24 V, sin embargo según los análisis en 5.2.4 (Tabla 5.3) hay una caída de voltaje de 2.02 V por lo que se considerará que la perturbación tenga un valor de:

$$n_{(k)} = \Delta V \cdot \delta = -2.02 \cdot 0.4363 = -0.8813 \qquad (5.25)$$

Este dato se ingresa en forma de escalón mediante el bloque denominado N_z tal como se muestra en la figura 5.27.

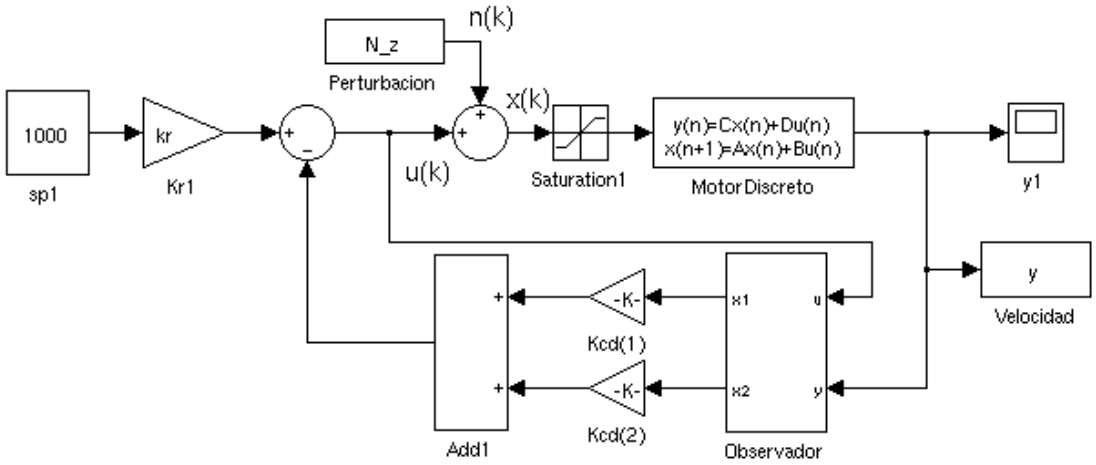


Fig.5.27 Sistema con entrada de referencia y perturbación en la entrada de la Planta

El resultado de la simulación se muestra en la figura 5.30 y como puede observarse este presenta un comportamiento similar al obtenido con un motor DC real (figura 5.24), lo

cual comprueba la hipótesis de que el error en estado estable en este caso es causado por la caída de tensión en el puente H y la fuente de alimentación.

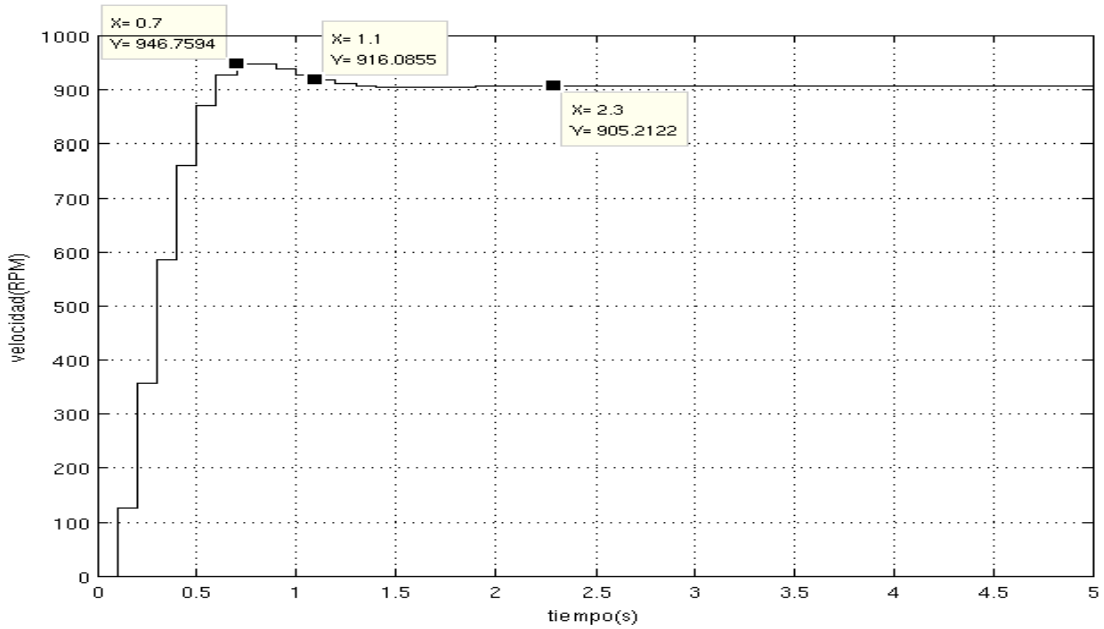


Fig.5.28 Efecto de la perturbación en la salida para un set point de 1000 RPM

Para una perturbación de 1 V (entrada escalón) y un set point de 0 RPM se esperaría según la ecuación 5.3 que la salida sea de 95.5 RPM, sin embargo la realimentación de la salida para la estimación de estados ayuda a reducir este efecto siendo la salida de 59 RPM en estado estable tal como se muestra en la figura 5.29.

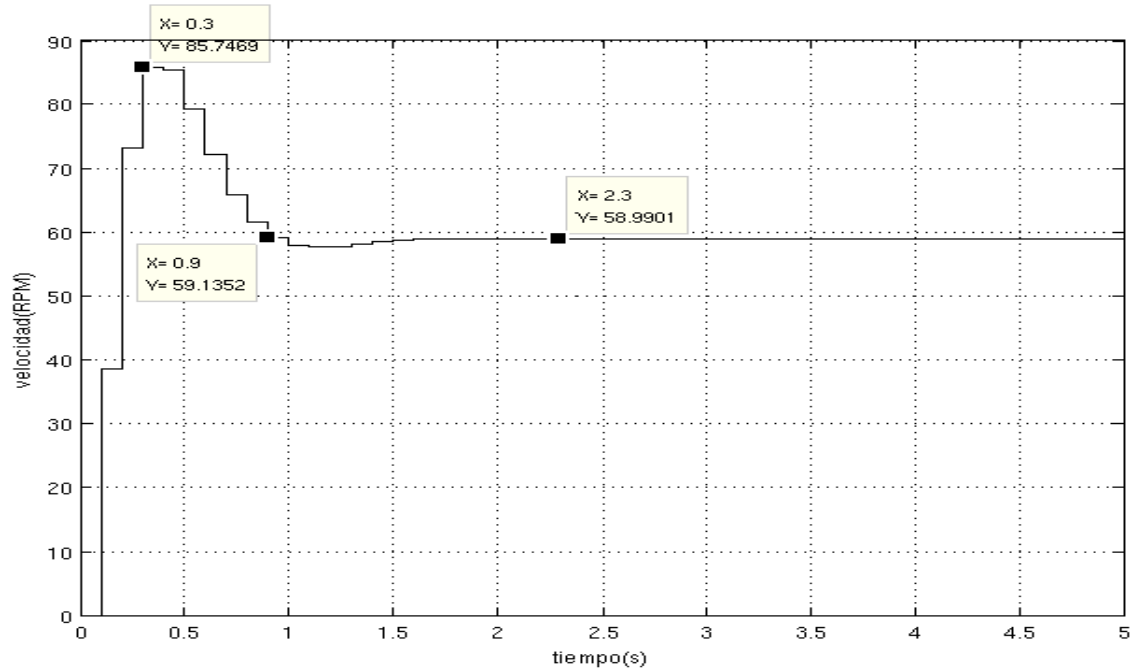


Fig.5.29 Efecto de la perturbación en la salida para un set point de 0 RPM

b) Perturbación a la salida de la planta

Este tipo de perturbación es aquel que altera la salida de la planta que se quiere controlar, por lo que en este caso la perturbación tendrá unidades de velocidad a diferencia de la perturbación considerada en la entrada la cual tiene unidades de voltaje.

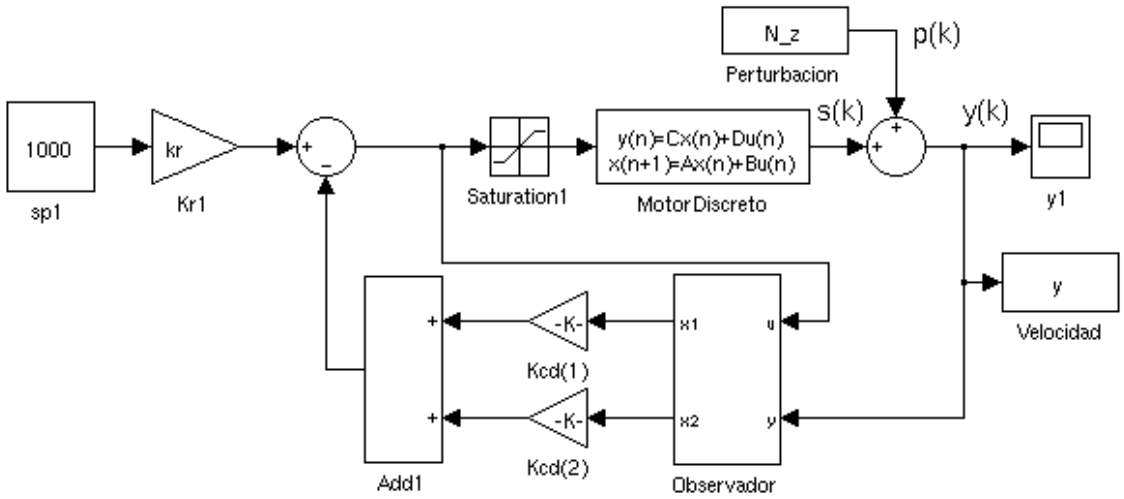


Fig.5.30 Sistema con entrada de referencia y perturbación en la salida de la Planta

Considerando una perturbación tipo escalón unitario la salida es mostrada en la figura 5.31, y presenta un efecto indeseado producto de la perturbación que no es eliminado en su totalidad.

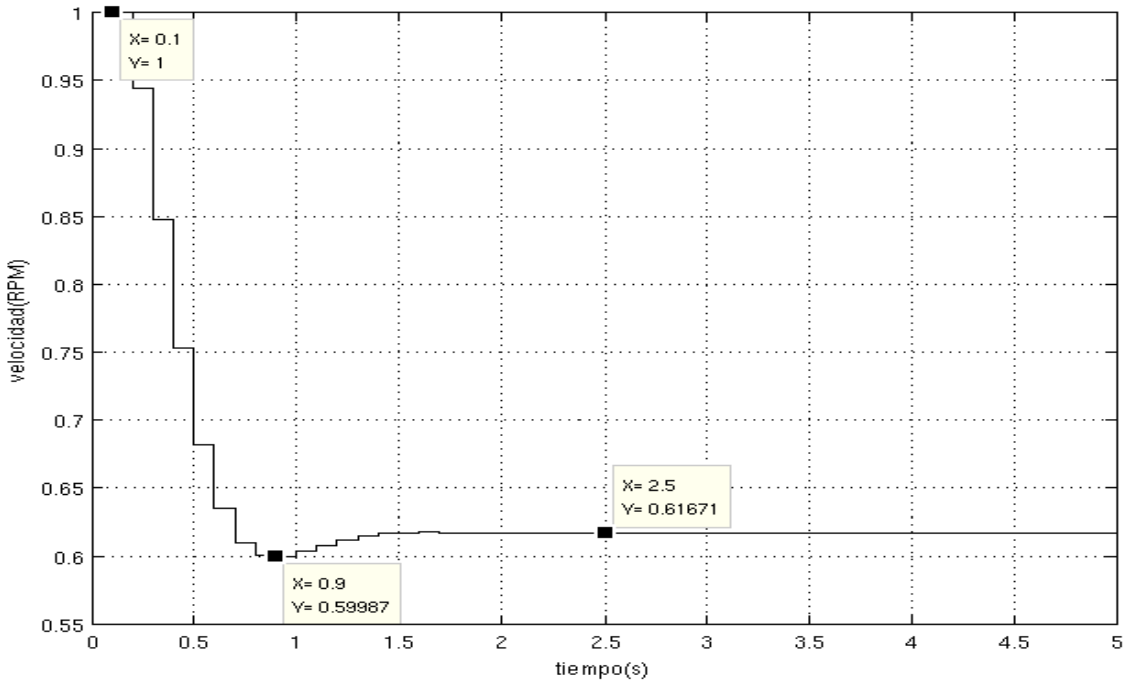


Fig.5.31 Efecto de la perturbación en la salida de la Planta

5.3. Control de un motor DC en tiempo real basado en un sistema de seguimiento

Un control con entrada de referencia permite establecer los polos del sistema en valores adecuados para una respuesta transitoria aceptable, sin embargo este tipo de control no calcula la señal de control para que la salida siga a la entrada sino que se limita a calcular valores para la señal de control que pondría a la salida en su valor esperado bajo condiciones ideales (fuente de alimentación ideal y motor sin carga).

Sin embargo en muchos sistemas de control es necesario que la salida siga a la entrada de manera que si la fuente de alimentación variase un poco su valor o si la carga del motor aumentase entonces el controlador aumente la potencia entregada y así se logre que la salida mantenga valores próximos a la referencia.

Es así que la solución es la implementación de un control basado en un sistema de seguimiento en el que la salida se realimente no sólo para la estimación de estados sino también para la comparación con la referencia por lo que en este tipo de sistemas es necesaria la implementación de uno o dos integradores que eliminen el error en estado estable si es que la planta no los posee.

5.3.1. Diseño de un observador de tipo predictivo con oscilaciones muertas

En 5.1.3 se diseñó un observador completo de tipo predictivo de manera que sus polos en lazo cerrado permitan una respuesta 3 veces más rápida que el sistema en lazo cerrado a controlar.

Entre las múltiples ventajas de un control en tiempo discreto destaca la posibilidad de implementar un observador completo de tipo predictivo con oscilaciones muertas de manera que en 2 periodos de muestreo (el orden de nuestro sistema es 2) el error en el observador sea cero por lo que será usado [35].

Con las matrices *G* y *H* de la ecuación 5.9 y la matriz *C* de la ecuación 5.5 obtenemos la matriz de controlabilidad *Mo* cuyo rango coincide con el orden del sistema de lo cual se desprende que el sistema es observable.

```
>> Mo = [C', G'*C']

Mo =

   1.0e+04 *

         0    0.0459
    1.8350    1.0954

>> no = rank (Mo)

no =

     2
```

Fig.5.32 Salida de MATLAB – Matriz de Observabilidad y rango

Para el cálculo de la ganancia del observador hacemos uso de la fórmula de Ackerman [35] como puede verse en la figura 5.32:

```
>> p=[1 0 0]

p =

     1     0     0

>> PHY=polyvalm(p,G)

PHY =

   -0.1109   -2.4038
    0.0125    0.2362

>> Kod=PHY*(inv(Mo'))*[0;1]

Kod =

   1.0e-03 *

   -0.2415
    0.0273
```

Fig.5.33 Salida de MATLAB – Ganancia del observador con oscilaciones muertas

$$K_{od} = [-0.000010887 \quad 0.000016886] \qquad (5.26)$$

5.3.2. Diseño del controlador realimentación con realimentación de estados e integrador en trayectoria directa

Dado que en el apartado C de 5.1.2 se determinó los polos en lazo cerrado que debería tener el sistema para una respuesta transitoria con las características ahí definidas ese análisis no se volverá a hacer de manera que se diseñará para que nuestro sistema tenga dichos polos.

Sin embargo la inclusión de un integrador incrementa en uno el orden del sistema por lo que debe especificarse otro polo que tenga la propiedad deseable de no alterar los efectos de los otros dos polos complejos conjugados.

Es conocido en la teoría de control moderno en tiempo continuo que cuando se desea incluir uno o más polo que no afecten a otros dos polos (cuyas características son deseables

en el sistema) estos deben tener su parte real lo más alejada de la parte real de los otros dos polos complejos conjugados, esto es, si se pudiera los otros polos deben tener su parte real en el menos infinito [36].

Lo anteriormente dicho se traduce a la teoría de control moderno en tiempo discreto localizando los polos en el origen, de esta manera tenemos que los polos en lazo cerrado deben ser:

$$\begin{aligned} p_{d1} &= -0.609966 + j0.268162 \\ p_{d2} &= -0.609966 - j0.268162 \\ p_{d3} &= 0 \end{aligned} \tag{5.27}$$

Estos polos nos dan el siguiente polinomio característico:

$$p = z^3 - 1.2199z^2 + 0.440z + 0 \tag{5.28}$$

Usando este polinomio característico y las matrices G y H de la ecuación 5.9 hacemos los siguientes cálculos en MATLAB.

```
>> p=[1 -1.2199 0.4440 0]

p =

    1.0000   -1.2199    0.4440         0

>> G1=[G H;0 0 0]

G1 =

   -0.0964   -4.8021    0.0250
    0.0250    0.5970    0.0021
         0         0         0

>> H1=[0;0;1]

H1 =

     0
     0
     1

>> M=[H1 G1*H1 G1^2*H1]

M =

         0    0.0250   -0.0125
         0    0.0021    0.0019
    1.0000         0         0
```

Fig.5.34 Salida de MATLAB – Cálculo de la matriz de controlabilidad

La matriz M es la matriz de controlabilidad ampliada y puede verificarse que su rango es 3. Siguiendo con la formula de Ackerman obtenemos:

```
>> PHY=polyvalm(p,G1)

PHY =

    0.0430   -0.1022    0.0185
    0.0005    0.0578   -0.0006
         0         0         0

>> K=[0 0 1]*(inv(M))*PHY

K =

   -1.0503   22.6481   -0.7193

>> N=[G-eye(2) H;C*G C*H]

N =

   1.0e+04 *

   -0.0001   -0.0005    0.0000
    0.0000   -0.0000    0.0000
    0.0459    1.0954    0.0039

>> K=(K+[0 0 1])*(inv(N))

K =

    3.1509   19.5698    0.0042
```

Fig.5.35 Salida de MATLAB – Cálculo de la ganancia del controlador

```
>> Kcd=[K(1),K(2)]

Kcd =

    3.1509   19.5698

>> Kid=K(3)

Kid =

    0.0042
```

Fig.5.36 Salida de MATLAB – Ganancia de la realimentación de estado y del integrador

$$K_{cd} = [3.1509 \quad 19.5698]$$
$$K_{id} = 0.0042 \tag{5.29}$$

5.3.3. Simulación del sistema

a) Simulación usando MATLAB

La simulación es realizada en SIMULINK de manera que son elaborados un script para generar los parámetros y un diagrama de bloques presentado en la figura 5.37.

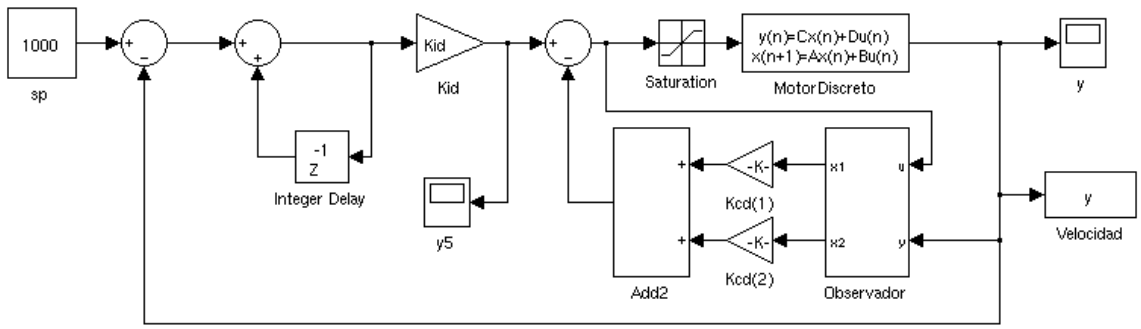


Fig.5.37 Diagrama de bloques del sistema de seguimiento

Puesto que el máximo sobrepico es 4.47%, y el tiempo de establecimiento es 1.1 s tal como se observa en la figura 5.38, se puede decir que el sistema tiene un desempeño que cumple con los requisitos impuestos en la respuesta transitoria

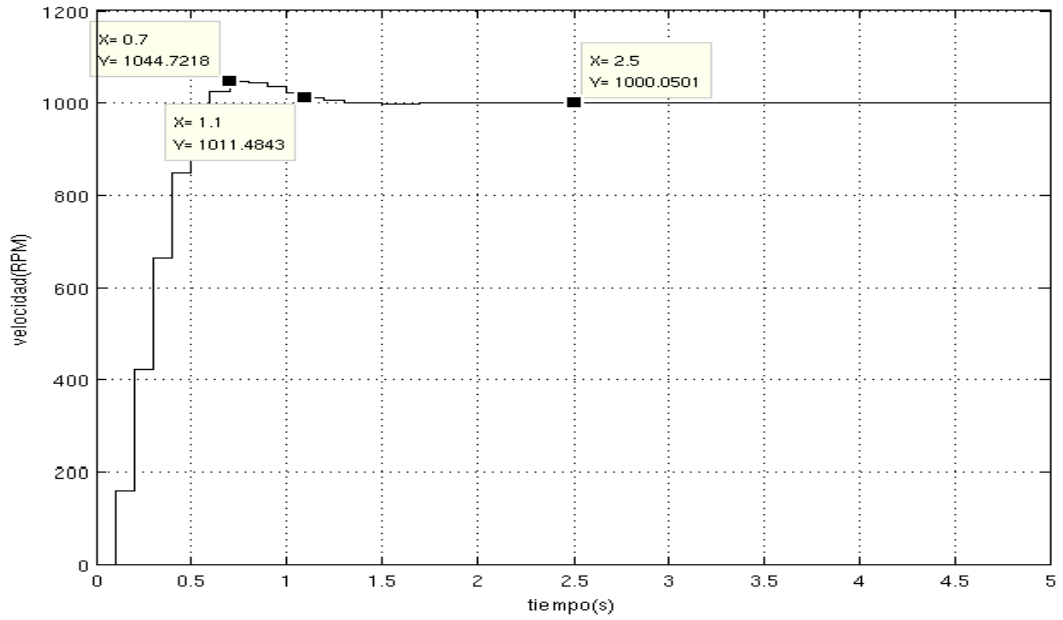


Fig.5.38 Resultados de la simulación para el sistema de seguimiento usando MATLAB

b) Simulación usando RTSJ

El resultado de la simulación para un set point de 1000 RPM con condiciones iniciales nulas se muestra en la figura 5.39.

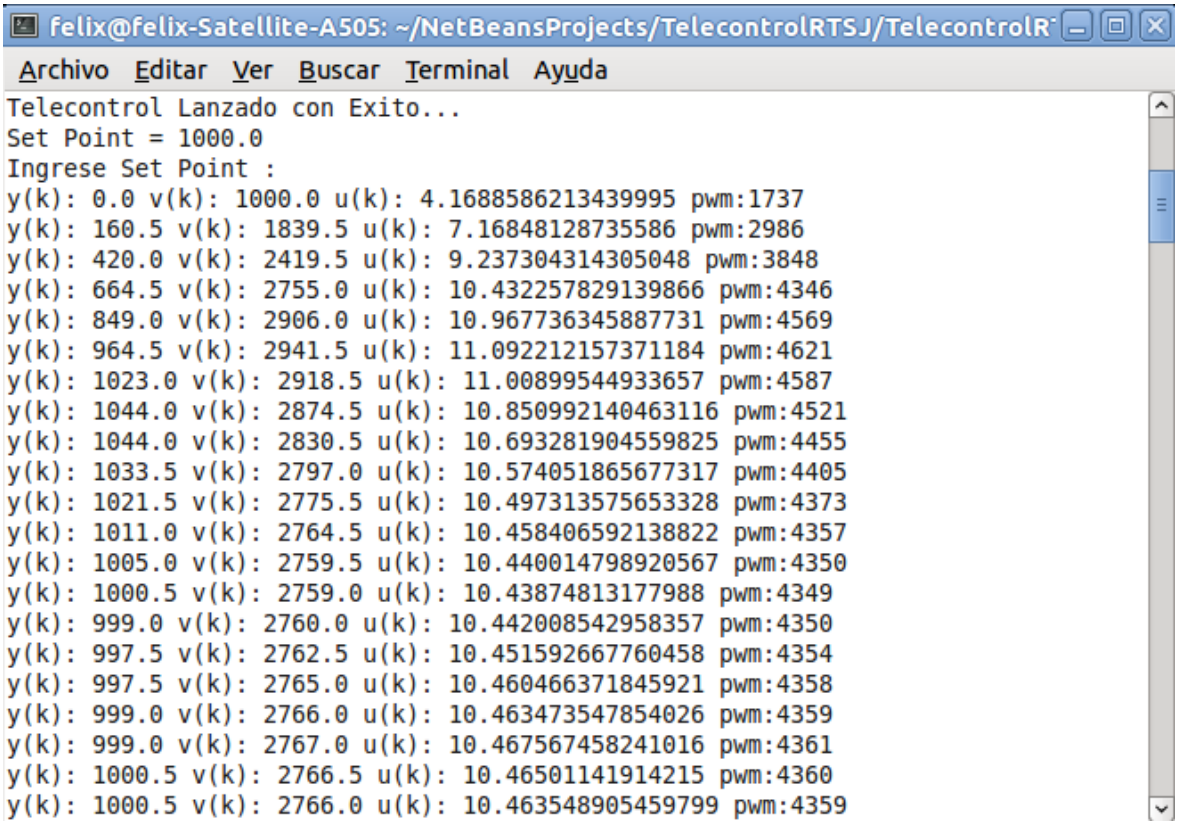
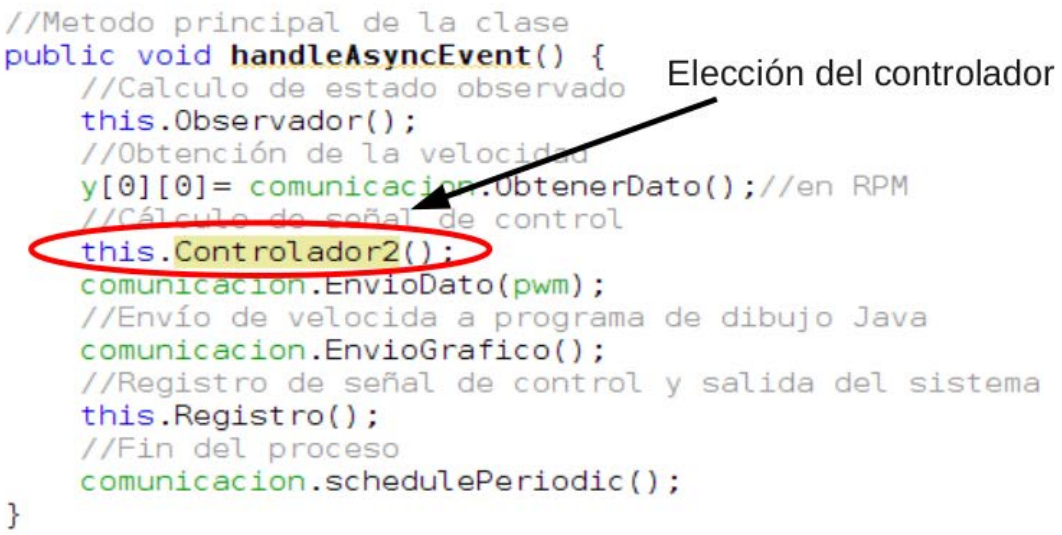


Fig.5.39 Resultado de la simulación usando TelecontrolRTSJ y SimulacionFPGA

El tiempo de establecimiento es de 1.1 s y con un máximo sobrepico de 4.40%, dado de que los resultados son satisfactorios se continúa con la prueba haciendo uso del FPGA y el motor DC real.

5.3.4. Prueba del sistema de seguimiento con integrador en trayectoria directa con un motor DC real

El programa *TelecontrolRTSJ* es modificado para elegir el método *Controlador2* tal como se muestra en la figura 5.40, además debe verificarse que la dirección IP apunte a la del FPGA.




```
//Metodo principal de la clase
public void handleAsyncEvent() {
    //Calculo de estado observado
    this.Observador();
    //Obtención de la velocidad
    y[0][0]= comunicacion.ObtenerDato();//en RPM
    //Cálculo de señal de control
    this.Controlador2();
    comunicacion.EnvioDato(pwm);
    //Envío de velocida a programa de dibujo Java
    comunicacion.EnvioGrafico();
    //Registro de señal de control y salida del sistema
    this.Registro();
    //Fin del proceso
    comunicacion.schedulePeriodic();
}
```

Fig.5.40 Elección del método Controlador2 (sistema de seguimiento)

El programa en el FPGA es el mismo que se muestra en el anexo D. La fuente de alimentación usada es de tipo **lineal**.

De las múltiples ejecuciones con diferentes set point se escogieron las que se muestran en las figuras 5.41, 5.42 y 5.43 que representan en promedio los máximos sobre picos y tiempo de establecimiento para condiciones iniciales nulas obtenidos en general.

Set Point (RPM)	Máximo Sobre Pico (%)	Tiempo de establecimiento (s)
1000	0.00	1.1
1500	0.00	1.1
2000	0.00	1.1

TABLA Nº 5.6 Respuesta transitoria usando un sistema de seguimiento para el control de velocidad de un motor real y para diferentes set point

Es fácil observar que aunque el controlador ha sido diseñado para que la salida presente una forma sobreamortiguada la respuesta ha sido subamortiguada, una posible causa de este hecho es que el programa *TelecontrolRTSJ* asume que la señal de control que envía se establece tal cual, sin embargo según se pudo observar en el sistema con entrada de referencia la señal de control establecida (voltaje promedio aplicado) es menor al que debe aplicarse debido a la disminución del voltaje de la fuente de alimentación y pérdidas en el puente H.

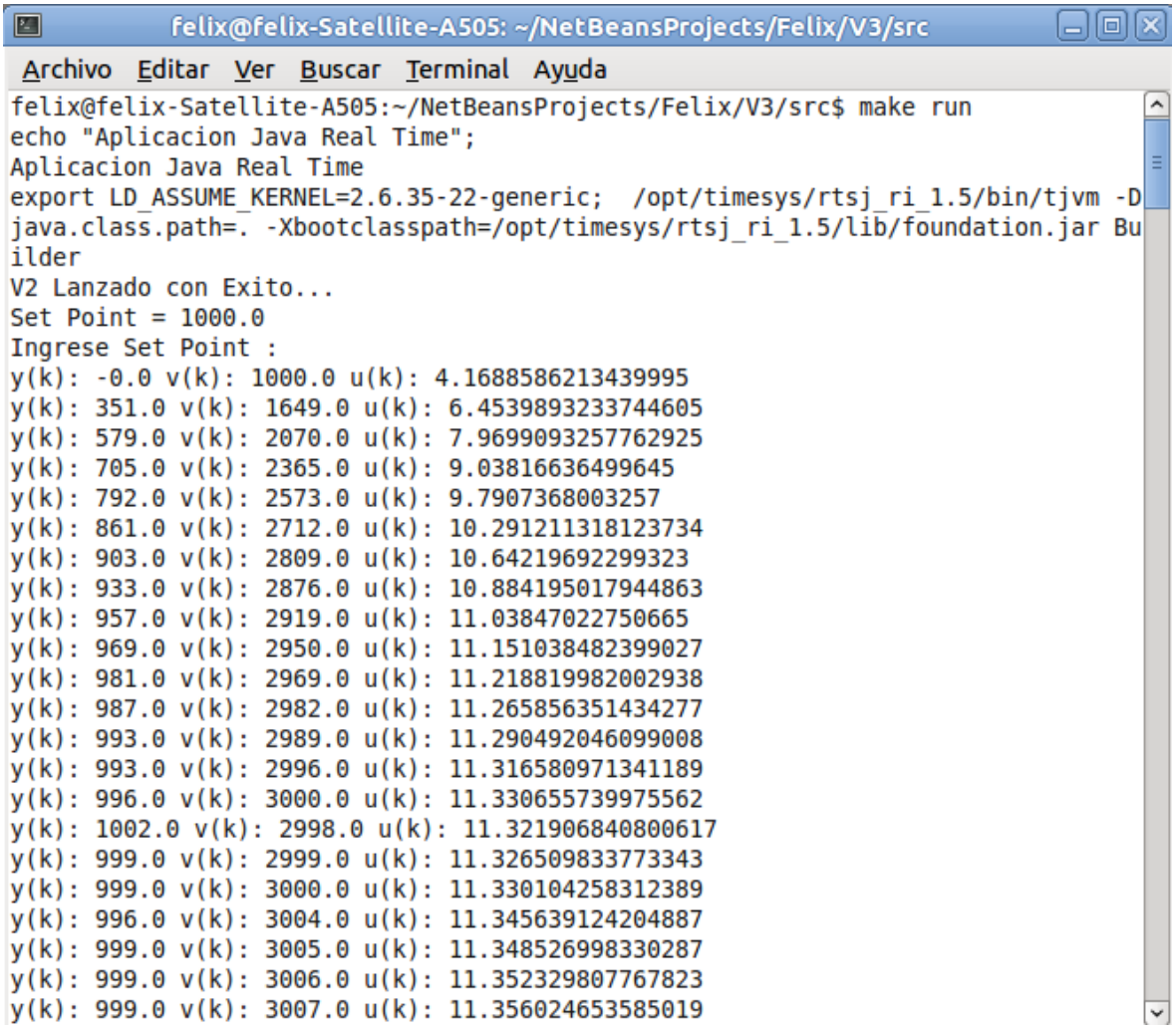


Fig.5.41 Control de la velocidad angular del eje del motor real con un set point de 1000 RPM usando un sistema de seguimiento

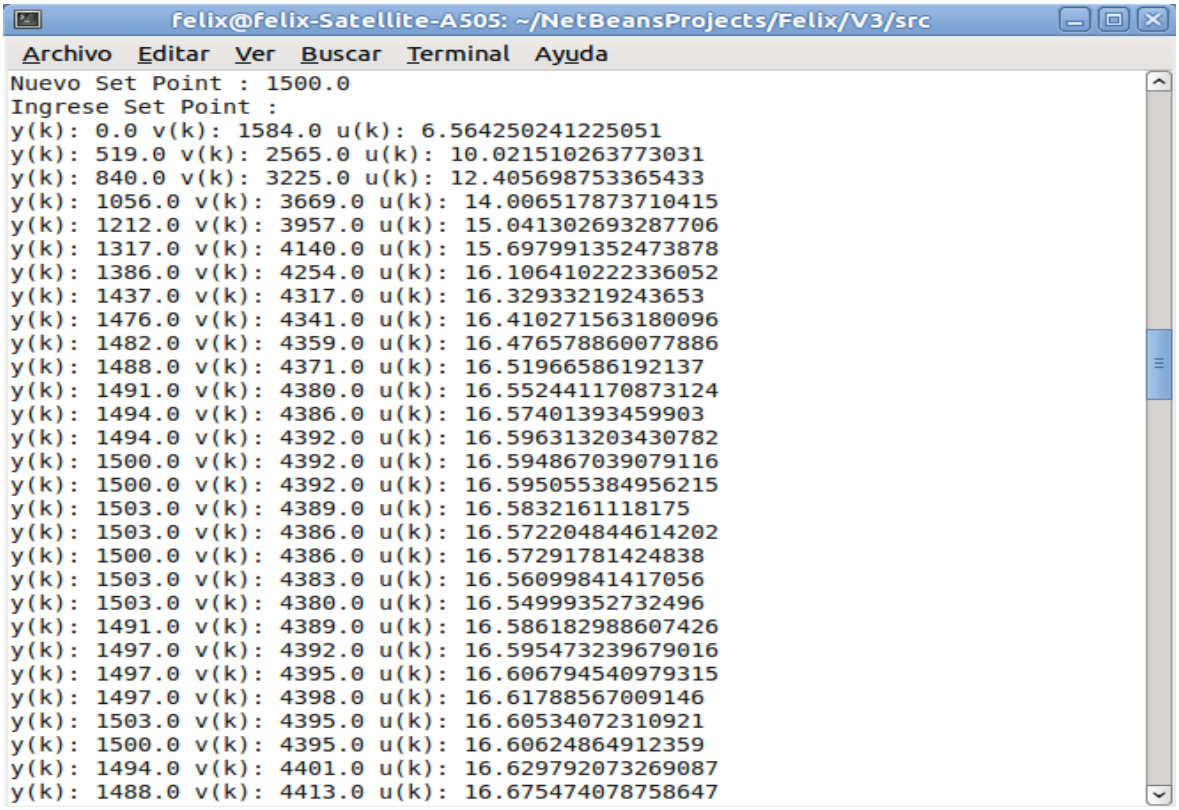


Fig.5.42 Control de la velocidad angular del eje del motor real con un set point de 1500 RPM usando un sistema de seguimiento

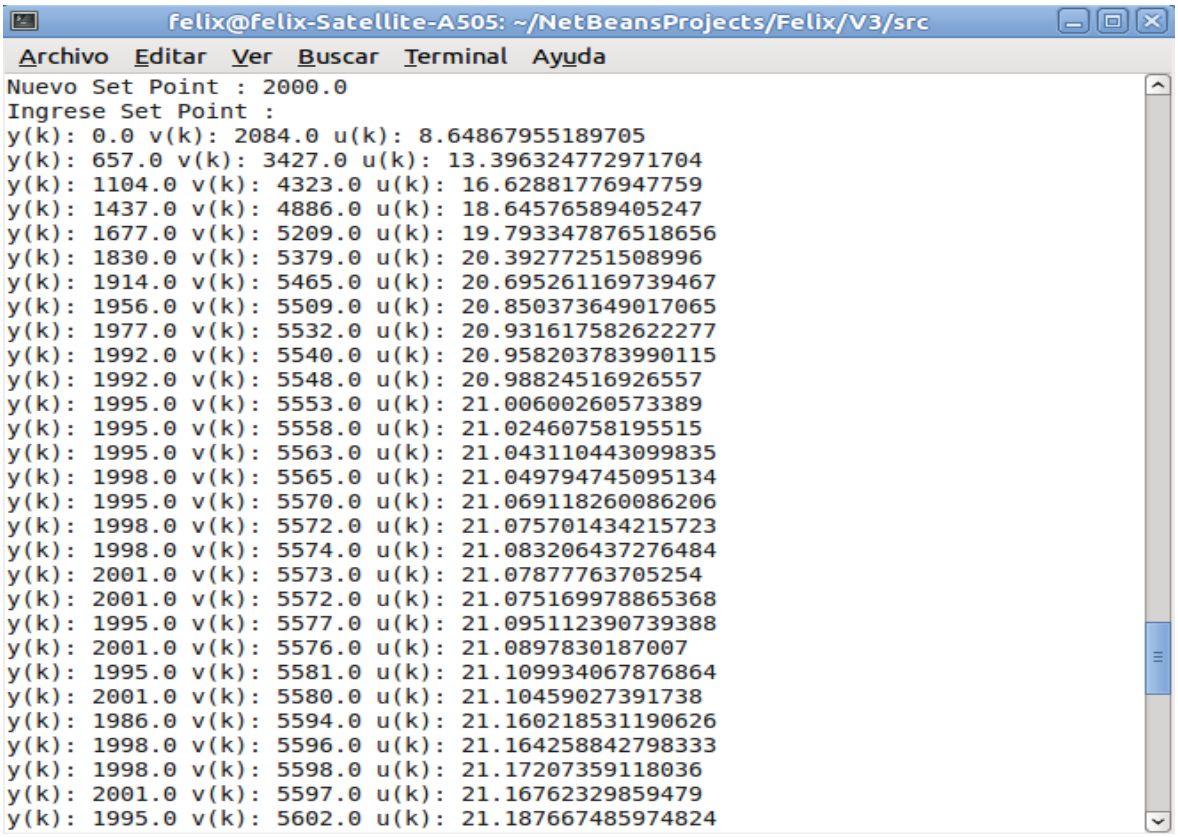


Fig.5.43 Control de la velocidad angular del eje del motor real con un set point de 2000RPM usando un sistema de seguimiento

5.3.5. Análisis de las perturbaciones en un sistema de seguimiento

a) Perturbación a la entrada de la planta

Suponiendo una caída de voltaje constante y considerando que el ciclo de trabajo aumenta hasta conseguir la salida deseada, la perturbación tendrá la forma de una función rampa tal como se muestra en la ecuación 5.25 .

$$n_{(k)} = \Delta V \cdot \delta = \Delta V \frac{u_{(k)}}{24} \qquad (5.30)$$

La figura 5.27 muestra el diagrama de bloques a usar para simular los efectos de la perturbación. Para poder comparar los resultados de la simulación con los obtenidos en el subcapítulo anterior se considerara un set point de 1000 RPM y un voltaje de alimentación de 24 V.

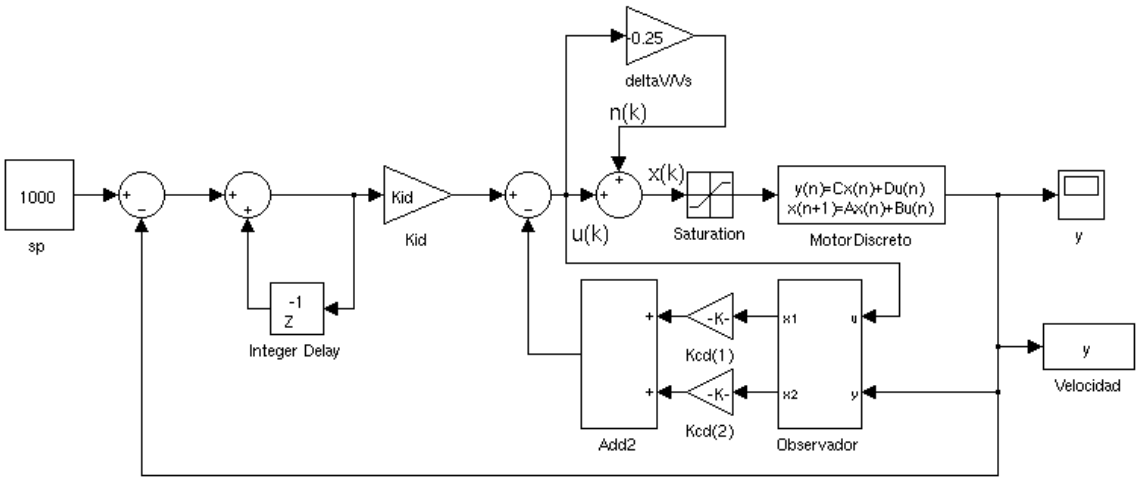


Fig.5.44 Sistema de seguimiento con una perturbación en la entrada de la planta

Para un valor ΔV/Vs=-0.25 se obtiene la salida mostrada en la figura 5.45 cuyo resultado es similar al resultado obtenido en 5.3.4 (sistema de seguimiento con un motor DC real) cuando se aplica un set point de 1000 RPM al sistema de seguimiento tal como se mostró en la 5.41, en el cual la velocidad muestra una forma subamortiguada a pesar de haberse diseñado para una respuesta sobreamortiguada. Nótese que en este caso la caída de tensión es considerablemente alta ΔV=6.0 V, lo cual se debe al uso de una fuente de alimentación lineal con una pobre regulación de carga.

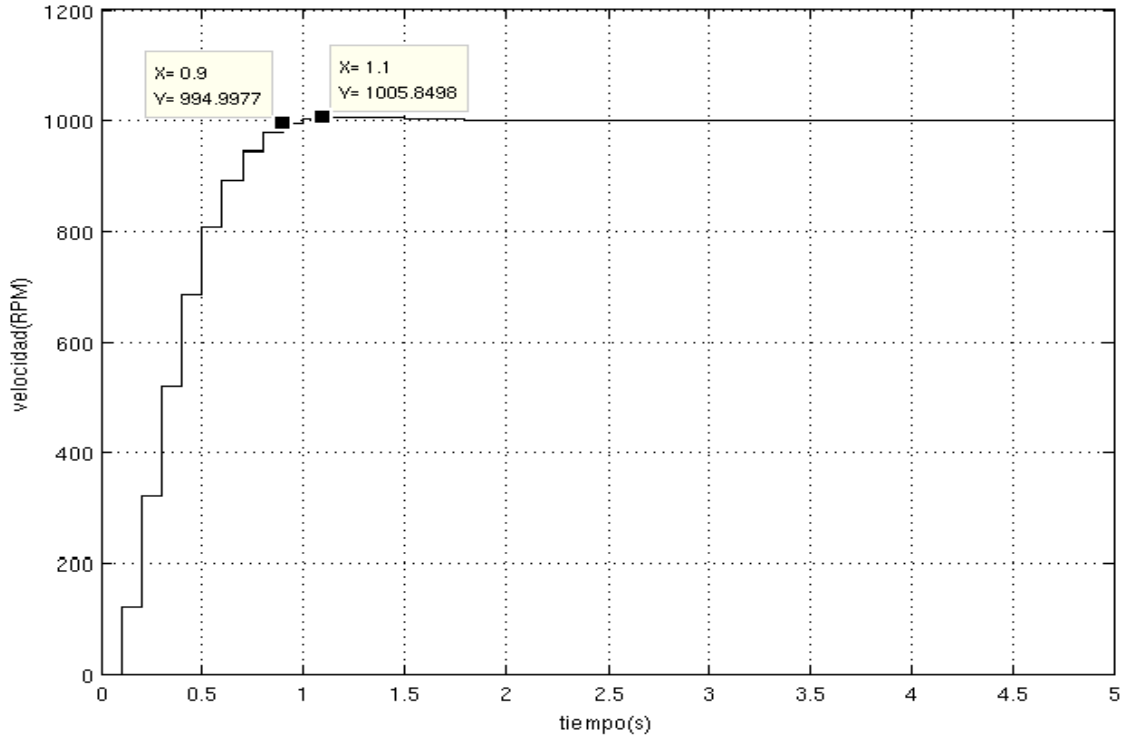


Fig.5.45 Efecto de la perturbación en la salida para un set point de 1000 RPM

En figura 5.46 se muestra la salida del sistema con un set point de 0 RPM y una perturbación tipo escalón unitario.

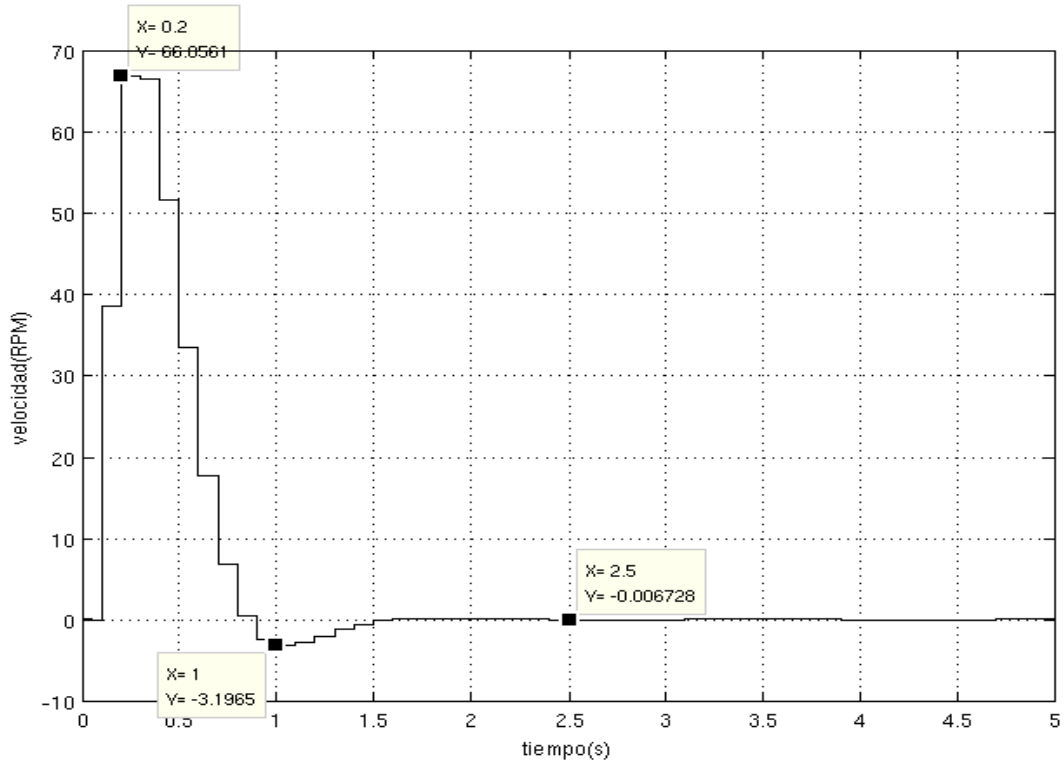


Fig.5.46 Efecto de la perturbación en la salida para un set point de 0 RPM

Cabe mencionar que según los resultados este sistema de seguimiento es más robusto con respecto a un sistema con entrada de referencia y esto puede comprobarse al comparar la figura 5.46 y la figura 5.29, en la cual esta última llega a un máximo pico de 85 RPM y luego decrece no hasta cero sino hasta 56 RPM.

b) Perturbación a la salida de la planta

Este tipo de perturbación es aquel que altera la salida de la planta que se quiere controlar.

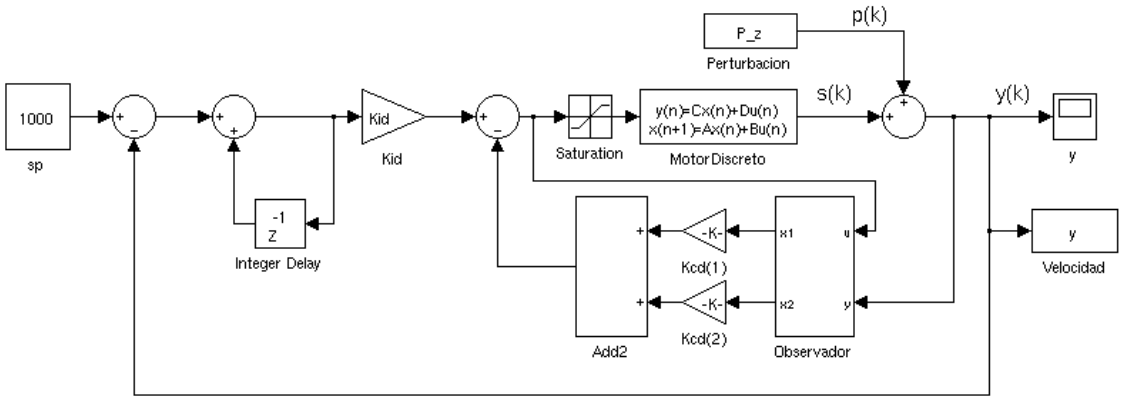


Fig.5.47 Sistema de seguimiento con una perturbación en la salida de la Planta

Considerando una perturbación tipo escalón unitario, la salida se muestra en la figura 5.46. Como puede observarse, el efecto indeseado de la perturbación es eliminado en su totalidad dado que este sistema es de seguimiento.

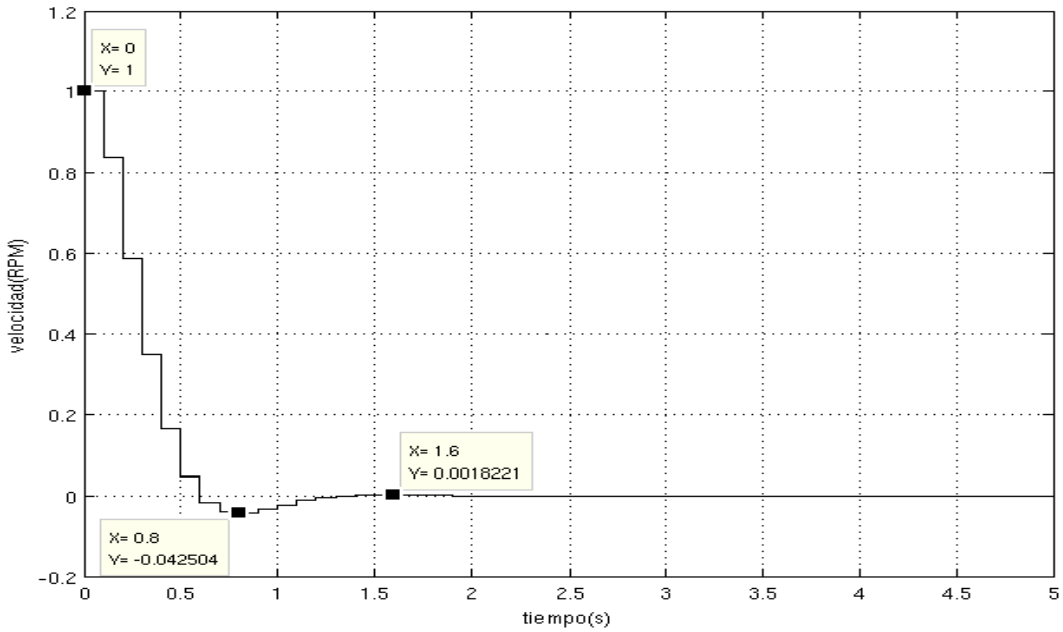


Fig.5.48 Efecto de la perturbación en la salida para un set point de 1000 RPM

5.3.6. Pérdida de paquetes en la red

Como se pudo observar en el apartado b) de 5.2.3, la pérdida de un dato debido a la ausencia de sincronización genera sobrepicos mayores al esperado y en general afecta la precisión del control lo cual se solucionó con la implementación de la clase SincronizacionUDP.

Sin embargo la pérdida de datos también puede deberse a las pérdidas de paquetes propios de una red y el sistema debe ser capaz de ser lo más inmune posible a estas.

A los paquetes que contienen la variable manipulada (voltaje) y que por tanto es enviado por la PC se denominará paquete_u y a los paquetes que contienen la variable regulada (velocidad angular) y que por tanto es enviado por el FPGA se denominará paquete_y.

Haciendo uso de los programas *TelecontrolRTSJ* y SimulacionFPGA se puede simular la pérdida de paquetes tanto para paquetes tipo paquete_u y tipo paquete_y.

a) Pérdida de paquete_u

Para el sistema de seguimiento desarrollado en el capítulo 5.2 se obtuvo el siguiente cuadro comparativo para un set point de 1000 RPM.

Cantidad de paquetes perdidos	Máximo Sobrepico (%)	Tiempo de establecimiento criterio de 2% (s)
0 paquetes	4.44	1.1
1 paquete por cada 2	15.05	1.3
1 paquete por cada 3	12.05	1.2
1 paquete por cada 4	9.05	1.2
1 paquete por cada 5	9.5	1.2
1 paquete por cada 6	10.2	1.1
1 paquete por cada 7	9.05	1.2
1 paquete por cada 10	9.05	1.2

TABLA Nº 5.7 Pérdida de paquetes con destino FPGA y origen PC

b) Pérdida de paquete_y

Para el sistema de seguimiento desarrollado en el capítulo 5.2 se obtuvo el siguiente cuadro comparativo para un set point de 1000 RPM.

Cantidad de paquetes perdidos	Máximo Sobrepico (%)	Tiempo de establecimiento criterio de 2% (s)
0 paquetes	4.44	1.1
1 paquete por cada 2	0.00	1.5
1 paquete por cada 3	0.00	1.0
1 paquete por cada 4	0.00	0.8
1 paquete por cada 5	0.00	0.7
1 paquete por cada 6	1.55	0.9
1 paquete por cada 7	1.55	0.9
1 paquete por cada 10	5.00	1.1

TABLA Nº 5.8 Pérdida de paquetes con destino PC y origen FPGA

Estos resultados muestran que la pérdida de paquetes paquete_u afecta el sobrepico máximo esperado, mientras que la pérdida de paquetes paquete_y afecta el tiempo establecimiento esperado.

La pérdida de paquetes *paquete_u* se puede solucionar haciendo uso de un sistema de control predictivo de manera que el sistema de control (PC remota) pueda anticiparse y predecir situación futuras del proceso para así enviar no sólo la señal de control actual si no las que deben establecerse en futuros intervalos de muestreo [1].

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. La medición de velocidad realizando una cuenta de pulsos en una ventana de tiempo resultó superior al esquema de medir el intervalo de tiempo entre los flancos de subida tal como puede observarse al comparar la figura 3.8 (intervalo entre flancos de subida) con la figura 3.10 (cuenta de pulsos).
2. Se había previsto la implementación en el FPGA del protocolo Ethernet, IP y UDP mas no ARP, sin embargo resultó necesaria la implementación de este último ya que aunque al FPGA se le daba la MAC y la IP de la PC en tiempo de compilación resulta que no se podía hacer lo mismo en el caso de la PC, es decir, se le podía dar la IP del FPGA mas no su MAC.
3. El uso de un mecanismo de interrupción para generar los periodos de muestreo en el FPGA ha resultado ser una mejor opción con respecto al uso de bucles *while* o *for* ya que se hace un mejor aprovechamiento de los recursos del procesador MicroBlaze.
4. El prototipo desarrollado ha permitido hacer la identificación de parámetros de un motor DC de manera satisfactoria, tal como puede verse en la figura 5.15, además de permitir el desarrollo de distintos esquemas de control.
5. La implementación de la clase *SincronizacionUDP* en *TelecontrolRTSJ* y su similar en el FPGA ha permitido solucionar los problemas de sincronización que afectaban de manera negativa al proceso de control (sobrepicos mayores a los esperados) tal y como puede comprobarse al comparar la figura 5.19 (sin *SincronizacionUDP*) y la figura 5.20 (con *SincronizacionUDP*).
6. La pérdida de paquetes que contienen la señal de control (paquete_u) afecta de manera negativa al sobrepico máximo esperado más que al tiempo de establecimiento tal como puede comprobarse al observar la tabla 5.7.

7. La pérdida de paquetes que contienen la información del sensor (paquete_y) afecta de manera negativa el tiempo de establecimiento esperado más que al sobrepico máximo tal como puede comprobarse al observar la tabla 5.8.
8. La implementación de un sistema de seguimiento muestra una mayor robustez contra las perturbaciones internas (caída de tensión en el convertidor o una pobre regulación de carga en la fuente de alimentación) y las perturbaciones externas (carga en el eje del motor) tal como se puede observar comparando la figura 5.28 con la figura 5.45 y la figura 5.31 con la figura 5.48.
9. El sistema de seguimiento implementado ha dado resultados satisfactorios cuando se trabaja en una red pequeña (según clasificación IP la red del laboratorio del CID en el que se ha trabajado es clase C) la cual consistió en un switch y tres PC´s conectadas a la red de las cuales una de ellas ejecutaba los procesos RTSJ para el control.
10. El programa *GraficoJAVA*, el cual ha sido implementado en este trabajo, ha dado resultados satisfactorios al hacer una representación gráfica de la velocidad del eje del motor. Sin embargo este programa no está limitado a graficar velocidades si no que, por ejemplo, si el FPGA enviara la señal digitalizada de unos sensores de electrocardiograma o alguna otra señal entonces esta sería graficada.

RECOMENDACIONES

1. Es recomendable que la tecnología en la cual se base el puente H sea MOSFET, ya que en comparación con la tecnología BJT ofrece dos ventajas que para el proceso de modelamiento y de control son de importante interés. La primera ventaja es la alta velocidad de conmutación que intrínsecamente tiene un dispositivo basado en tecnología MOSFET los cuales se encuentran en un rango de cientos de miles de Hertz mientras que la tecnología BJT se encuentra en un rango de decenas de miles de Hertz. La segunda ventaja es la baja disipación de potencia en el dispositivo que por lo general es mucho menor de la potencia que disiparía un dispositivo BJT bajo las mismas condiciones.
2. Como adicional a este prototipo se recomienda la implementación del protocolo ICMP en el FPGA para así tener un mecanismo de control de los paquetes enviados además de poder hacer pruebas de conectividad (ping) cuando el FPGA opere en una red más grande.
3. Dado que la precisión en el modelamiento y el control de un motor DC depende entre otras variables de la estabilidad del voltaje de la fuente DC es recomendable usar fuentes conmutadas ya que por lo general tienen una mejor regulación voltaje en comparación con las fuentes lineales.
4. Se recomienda la reproducción de este prototipo para su uso en los laboratorios de electrónica relacionados al curso de control ya que con este es posible realizar tanto el modelamiento como el control de un motor DC lo cual resulta instructivo para los estudiantes que conozcan la Teoría de Control y que deseen aplicarla a sistemas reales haciendo uso de esquemas PID, PI+D, control en el espacio de estados, control óptimo y control predictivo entre otros.
5. En futuros trabajos relacionados a esta línea de investigación se recomienda la implementación de un controlador Predictivo en RTSJ y soporte para el control predictivo en el FPGA, ya que tal como se ha visto la pérdida de paquetes en la red altera la precisión en el control y un controlador predictivo tendría la capacidad de anticipar señales de control que deben establecerse para un determinado set point. Esto es un controlador predictivo podría enviar 10 paquetes donde el primer paquete correspondería a la señal de control que debe establecerse, y los nueve

paquetes restantes serían las señales de control que se establecerían en caso durante 9 intervalos de muestreo no llegase alguna señal de control.

6. Se recomienda la implementación de un puente H con una velocidad de conmutación mayor a 100 kHz, ya que permitiría hacer un control más preciso de la velocidad que el que se ha hecho, además de poder extender este control a un control de posición en donde se requiere señales de control más precisas. Siendo el principal factor limitante la velocidad de conmutación de los dispositivos BJT es recomendable que el diseño de un puente H de mayor velocidad se base en tecnología MOSFET. Así también dado que se trabajará con señales de frecuencia debe tomarse en cuenta los efectos capacitivos e inductivos que generan las pistas del circuito impreso por lo que el diseño de éste debe estar precedido de un estudio de los efectos de estas líneas a una frecuencia de 100kHz.

ANEXOS

ANEXO A

Código de programa TelecontrolRTSJ – Control Remoto en tiempo real

Builder.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import javax.realtime.*;
/**
 *
 * @author Félix
 */
public class Builder {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        PriorityScheduler planificador = PriorityScheduler.instance();
        Scheduler.setDefaultScheduler(planificador);
        SchedulingParameters scheduling =
        new PriorityParameters(planificador.getMaxPriority());
        ReleaseParameters release = new PeriodicParameters(
        new RelativeTime(),  // inicio
        new RelativeTime(100,0),// periodo
        null,                // costo
        new RelativeTime(100,0),// deadline
        null,                // manejador en caso de exceso de costo
        null);               // manejador en caso se exceda deadline
        //Creación de hilo obtencion
        RealtimeThread obtencion= new ObtencionSP(scheduling, release);
        //Adicion de hilo a pila del planificador
        obtencion.setScheduler(planificador);
        //Creacion de hilo comunicacion
        RealtimeThread comunicacion=
        new ComunicacionUDP(scheduling, release,(ObtencionSP)obtencion);
        //Adicion de hilo a pila del planificador
        comunicacion.setScheduler(planificador);
        SincronizacionUDP sincronizacion=new SincronizacionUDP();
```

```
            sincronizacion.ReconocimientoTx();
            if (planificador.isFeasible()){
                System.out.println("Telecontrol Lanzado con Exito...");
                comunicacion.start();
                obtencion.start();
            }
            try{
    comunicacion.join();
                obtencion.join();
            }catch(Exception e){e.printStackTrace();}
    }
    }
```

ComunicacionUDP.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import javax.realtime.*;
import java.io.*;
import java.net.*;
/**
 *
 * @author Félix
 */
public class ComunicacionUDP extends RealtimeThread{

    volatile private byte[] buffer_tx = new byte[1024];
    volatile private byte[] buffer_rx = new byte[1024];

    private String IPfpga = "200.37.46.180";

    private double velocidad;
    private double sp;
    private String data;
private final double factor = 3.0;//(60/400)*(1/VentanaTiempo)
    private ObtencionSP obtencion;

public ComunicacionUDP(SchedulingParameters sched,
    ReleaseParameters release,ObtencionSP OSP){
        super(sched, release);
        obtencion=OSP;
    }

    public void run(){
        RealtimeThread hiloactual = currentRealtimeThread();
Controlador controlador = new Controlador(this);
AsyncEvent flag = new AsyncEvent();
        flag.addHandler(controlador);
       while(true){
            do{
                try {
                DatagramPacket datagrama_rx =
```

```
                new DatagramPacket(buffer_rx, buffer_rx.length);
                DatagramSocket socket_rx = new DatagramSocket(4405);
                socket_rx.receive(datagrama_rx);
                socket_rx.close();
                String cadena =
                new String(datagrama_rx.getData(),0,datagrama_rx.getLength());
velocidad = factor*Double.parseDouble(cadena);//en RPM
}
                catch (IOException ioe){
                    System.out.println("IO Error Rx: " + ioe);
}
                flag.fire();
            }while(hiloactual.waitForNextPeriod());
        }
    }

    public void EnvioDato(int pwm){
try {
//Envio de Señal de Control al FPGA
data=String.valueOf(pwm);
            buffer_tx = data.getBytes();
            InetAddress destino = InetAddress.getByName(IPfpga);
            DatagramPacket datagrama_tx =
            new DatagramPacket(buffer_tx,buffer_tx.length,destino,4406);
            DatagramSocket socket_tx = new DatagramSocket();
            socket_tx.send(datagrama_tx);
            socket_tx.close();

        }catch (IOException ioe){System.out.println("IO Error Tx: " + ioe);}
return;
    }

    public void EnvioGrafico(){
        try {
//Envio de magnitud y direccion de velocidad al graficador
data=String.valueOf(velocidad);
            buffer_tx = data.getBytes();
            InetAddress destino_g = InetAddress.getByName("localhost");
            DatagramPacket datagrama_tx_g =
            new DatagramPacket(buffer_tx,buffer_tx.length,destino_g,4407);
            DatagramSocket socket_tx_g = new DatagramSocket();
            socket_tx_g.send(datagrama_tx_g);
socket_tx_g.close();

        }catch (IOException ioe){System.out.println("IO Error Tx: " + ioe);}
return;
    }

    public double ObtenerDato(){
        return velocidad;
}

    public double ObtenerSP(){
        sp=obtencion.ObtenerSP();
return sp;
    }
}
```

Controlador.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import java.io.*;
import javax.realtime.*;
/**
 *
 * @author Félix
 */
public class Controlador extends AsyncEventHandler {
    //Vector de salida, vector de estados y vector de control
    private double[][] y={{0}};
    private double[][] x={{0},{0}};
    private double[][] u={{0}};
    private double[][] v={{0}};
    private double[][] sp={{0}};
    //Almacenamiento Temporal
    private double[][] Ax={{0},{0}};
    private double[][] By={{0},{0}};
    private double[][] Hu={{0},{0}};
    private double[][] Kiv={{0}};
    private double[][] Kcx={{0}};
    //Almacenamiento temporal
    private double cmd=0.0;
    private int pwm=0;
    private ComunicacionUDP comunicacion;
    private BufferedWriter bw;
    private PrintWriter salida;
    //Constructor de la clase
    public Controlador (ComunicacionUDP c) {
        super(
        new
PriorityParameters(PriorityScheduler.instance().getMinPriority()+10)
        ,null,null,null,null,null);
        comunicacion=c;
    }
    //Método principal de la clase
    public void handleAsyncEvent() {
        //Cálculo de estado observado
        this.Observador1();
        //Obtención de la velocidad
        y[0][0]= comunicacion.ObtenerDato();//en RPM
```

```
            //Cálculo de señal de control
            this.Controlador2();
            //Envío de trama UDP con el comando PWM que debe establecerse
            comunicacion.EnvioDato(pwm);
            //Envío de velocidad a programa de dibujo Java
            comunicacion.EnvioGrafico();
            //Registro de señal de control y salida del sistema
            this.Registro();
            //Fin del proceso
            comunicacion.schedulePeriodic();
        }

        private synchronized void Observador1(){
            double[][] Aod ={{-0.096401728260098,-0.371316174497830},
                             {0.025027978444791, 0.096401728260098}};
            double[][] Bod ={{-0.000241462256601743},{0.000027278450670338}};
            double[][] H ={{0.025027978444791},{0.002100568895341}};
            Matriz.Mult(Aod, 2, 2, x, 2, 1, Ax);//Ax=Aod*x
            Matriz.Mult(Bod, 2, 1, y, 1, 1, By);//By=Bod*y
            Matriz.Mult(H, 2, 1, u, 1, 1, Hu);//Hu=H*u
            Matriz.Suma(Ax, 2, 1, By, 2, 1, x);//x=Ax+Bu
            Matriz.Suma(x, 2, 1, Hu, 2, 1, x);//x=Ax+Bu
            return;
        }

        private synchronized void Observador2(){
            double[][] Aod ={{-0.0000353146471443884,-0.00413598454640689},
                             {0.00000030153021340858,0.0000353146471443884 }};
            double[][]                          Bod                          ={{-
0.000000000540454881555211},{0.000000000004425673701371 82}};
            double[][] H ={{0.0000000138144893156},{0.0000400826088374705}};
            Matriz.Mult(Aod, 2, 2, x, 2, 1, Ax);//Ax=Aod*x
            Matriz.Mult(Bod, 2, 1, y, 1, 1, By);//By=Bod*y
            Matriz.Mult(H, 2, 1, u, 1, 1, Hu);//Hu=H*u
            Matriz.Suma(Ax, 2, 1, By, 2, 1, x);//x=Ax+Bu
            Matriz.Suma(x, 2, 1, Hu, 2, 1, x);//x=Ax+Bu
            return;
        }

        private synchronized void Modelamiento1(){
            sp[0][0]=comunicacion.ObtenerSP();
            //Señal de control
            u[0][0]=sp[0][0];
            //Saturador para señal de control
            if (u[0][0]>1000) u[0][0]=1000;
```

```
            if (u[0][0]<-1000)u[0][0]=-1000;
            //Escala del comando pwm
            cmd=u[0][0];
            pwm=(int)cmd;
            System.out.println("y(k): "+y[0][0]+" u(k): "+u[0][0]);
            return;
        }

        private synchronized void Modelamiento2(){

            sp[0][0]=comunicacion.ObtenerSP();
            //Señal de control
            u[0][0]=sp[0][0];
            //Saturador para señal de control
            if (u[0][0]>1000) u[0][0]=1000;
            if (u[0][0]<-1000)u[0][0]=-1000;
            //Escala del comando pwm
            cmd=u[0][0];
            pwm=(int)cmd;
            System.out.println("y(k): "+y[0][0]+" u(k): "+u[0][0]);
            return;
        }

        private synchronized void Controlador1(){
            double kr=0.004167709;
            double[][] Kcd ={{-19.0578,-115.3941}};
            sp[0][0]=comunicacion.ObtenerSP();
            //Señal de control
            Matriz.Mult(Kcd, 1, 2, x, 2, 1, Kcx);//Kcd=Kcd*x
            u[0][0]=sp[0][0]*kr-Kcx[0][0];
            //Saturador para señal de control
            if (u[0][0]>24) u[0][0]=24;
            if (u[0][0]<-24)u[0][0]=-24;
            //Escala del comando pwm
            cmd=u[0][0]*(416.67);//10000/24
            pwm=(int)cmd;
            System.out.println(
                    "y(k):    "+y[0][0]+"    u(k):    "+u[0][0]+"    x1:"+x[0][0]+"
x2:"+x[1][0]
            );
            return;
        }

        private synchronized void Controlador2(){
            double[][] Kcd ={{3.15092767256869,19.5698095769503}};
```

```
        double[][] Kid ={{0.0041688586213442}};
        //Ecuación de Integrador
        sp[0][0]=comunicacion.ObtenerSP();
        v[0][0]=v[0][0]+sp[0][0]-y[0][0];
        //Señal de control
        Matriz.Mult(Kid, 1, 1, v, 1, 1, Kiv);//Kiv=Kid*v
        Matriz.Mult(Kcd, 1, 2, x, 2, 1, Kcx);//Kcd=Kcd*x
        u[0][0]=Kiv[0][0]-Kcx[0][0];
        //Saturador para señal de control
        if (u[0][0]>5) u[0][0]=5;
        if (u[0][0]<-5)u[0][0]=-5;
        //Saturador para señal del integrador
        if (v[0][0]>1923) v[0][0]=1923;
        if (v[0][0]<-1923)v[0][0]=-1923;
        //Escala del comando pwm
        cmd=u[0][0]*(416.67);//10000/24
        pwm=(int)cmd;
        System.out.println(
                "y(k): "+y[0][0]+" v(k): " +v[0][0]+
                " u(k): "+u[0][0]+" pwm:"+pwm
        );
        return;
    }
     private synchronized void Registro(){
        //Impresión de parámetros en archivo para posterior análisis
        try{
            bw = new BufferedWriter(new FileWriter("salida.dat", true));
            salida = new PrintWriter(bw);
            salida.println(u[0][0]+" "+y[0][0]);
            salida.close();
        }
        catch(IOException e){
            e.printStackTrace();
        }

        return;
    }
}
```

ObtencionSP.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import javax.realtime.*;
import java.io.*;
/**
 *
 * @author Félix
 */
public class ObtencionSP extends RealtimeThread{

    private double sp=1000;
    private InputStreamReader isr;
    private BufferedReader br;
    private String tecla="1";

    public ObtencionSP(SchedulingParameters sched, ReleaseParameters release){
        super(sched, release);
    }

    public void run(){
        RealtimeThread hiloactual = currentRealtimeThread();
        System.out.println("Set Point = " + sp);
       while(true){
            do{
                try{
                    System.out.println("Ingrese Set Point : ");
                    isr=new InputStreamReader(System.in);
                    br=new BufferedReader(isr);
                    tecla=br.readLine();
                    sp=Double.parseDouble(tecla);
                    System.out.println("Nuevo Set Point : "+sp);
                }
                catch(IOException ioe){
                    System.out.println("Error de entrada de Set Point");
                }
            }while(hiloactual.waitForNextPeriod());
        }
    }

    public synchronized double ObtenerSP(){
        return sp;
}

}
```

SincronizacionUDP

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import java.net.*;
import java.io.*;
/**
 *
 * @author Félix
 */
public class SincronizacionUDP {

    private static volatile byte[] buffer_rx= new byte[1024];
    private static volatile byte[] buffer_tx = new byte[1024];
    private String IPfpga = "200.37.46.180";
    private String data;
public SincronizacionUDP(){}

    public void ReconocimientoTx(){
        //Envio de Señal de requerimiento al FPGA
try{
            data="A";
            buffer_tx = data.getBytes();
            InetAddress destino = InetAddress.getByName(IPfpga);
            DatagramPacket datagrama_tx =
            new DatagramPacket(buffer_tx,buffer_tx.length,destino,4406);
            DatagramSocket socket_tx = new DatagramSocket();
            socket_tx.send(datagrama_tx);
            socket_tx.close();
        }
        catch(IOException e){
System.out.println("IO Error: " + e);
        }
return;
    }

    public int ReconocimientoRx(){
        int IndicadorInicio=0;
        //Recepcion de señal de reconocimiento desde la PC
try{
            DatagramSocket socket = new DatagramSocket(4406);
            DatagramPacket datagrama_rx =
            new DatagramPacket(buffer_rx,buffer_rx.length);
            socket.receive(datagrama_rx);
            String cadena =
            new String(datagrama_rx.getData(),0,datagrama_rx.getLength());
            if (cadena.equals("A")) IndicadorInicio = 1;
            socket.close();
}
        catch(IOException e){
            System.out.println("IO Error: " + e);
}
        return IndicadorInicio;
    }

}
```

Matriz.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Félix
 */

public class Matriz{

    public static void Suma(double[][] A,int DimX_A,int DimY_A,
    double[][] B, int DimX_B, int DimY_B,double[][] C){

        for (int i=0;i<DimX_A;i=i+1){
            for (int j=0;j<DimY_A;j=j+1){
C[i][j] = A[i][j]+B[i][j];
            }
        }
    }


    public static void Mult(double [][] A, int DimX_A, int DimY_A,
double [][] B, int DimX_B, int DimY_B,double[][] C){
double Temp;

        for (int i=0;i<DimX_A;i=i+1){
            for (int j=0;j<DimY_B;j=j+1){
                Temp = 0;
                for (int k=0;k<DimY_A;k=k+1){
                    Temp = Temp + A[i][k]*B[k][j];
                    C[i][j] = Temp;
                }
            }
        }
    }
}
```

ANEXO B

Código de programa SimulacionFPGA – Control Remoto en tiempo real

Builder.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import javax.realtime.*;
/**
 *
 * @author Félix
 */
public class Builder {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        PriorityScheduler planificador = PriorityScheduler.instance();
        Scheduler.setDefaultScheduler(planificador);
        SchedulingParameters scheduling =
        new PriorityParameters(planificador.getMinPriority());

        ReleaseParameters release = new PeriodicParameters(
        new RelativeTime(),      // inicio
        new RelativeTime(100,0),// periodo
null,              // costo
        new RelativeTime(100,0),// deadline
        null,                    // manejador en caso de exceso de costo
        null);                   // manejador en caso se exceda deadline

RealtimeThread recepcion=
        new RecepcionUDP(scheduling, release);
        recepcion.setScheduler(planificador);

        RealtimeThread comunicacion=
        new ComunicacionUDP(scheduling, release,(RecepcionUDP)recepcion);
        comunicacion.setScheduler(planificador);

        SincronizacionUDP sincronizacion=new SincronizacionUDP();
int indicador=0;
while(indicador==0){
            indicador=sincronizacion.ReconocimientoRx();
        }
        if (planificador.isFeasible()){
            System.out.println("F2 Lanzado con Exito...");
            recepcion.start();
```

```
            comunicacion.start();

        }
        try{
recepcion.join();
            comunicacion.join();
        }catch(Exception e){e.printStackTrace();}
    }

}
```

ComunicacionUDP.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import javax.realtime.*;
import java.io.*;
import java.net.*;
/**
 *
 * @author Félix
 */
public class ComunicacionUDP extends RealtimeThread{

    private byte[] buffer_tx = new byte[1024];
    private RecepcionUDP recepcion;
    private String data;

    public ComunicacionUDP(SchedulingParameters sched,
    ReleaseParameters release,RecepcionUDP RUDP){
        super(sched, release);
        recepcion = RUDP;

    }

    public void run(){
        RealtimeThread hiloactual = currentRealtimeThread();
Controlador controlador = new Controlador(this);
AsyncEvent flag = new AsyncEvent();
        flag.addHandler(controlador);
       while(true){
            do{
                flag.fire();
            }while(hiloactual.waitForNextPeriod());
        }
    }

    public void EnvioDato(int velocidad){
try {
            //Envio de velocidad a la PC
data=String.valueOf(velocidad);
            buffer_tx = data.getBytes();
            InetAddress destino = InetAddress.getByName("localhost");
            DatagramPacket datagrama_tx =
            new DatagramPacket(buffer_tx,buffer_tx.length,destino,4405);
```

```
            DatagramSocket socket_tx = new DatagramSocket();
            socket_tx.send(datagrama_tx);
            socket_tx.close();
        }catch (IOException ioe){System.out.println("IO Error: " + ioe);}
        return;
    }

    public double ObtenerDato(){
        return recepcion.ObtenerDato();
}
}
```

Controlador.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
import javax.realtime.*;
/**
 *
 * @author Félix
 */
public class Controlador extends AsyncEventHandler {
//Matrices de estado
    private double[][] A={{-0.096401,-4.802148},{0.0250279,0.596961}};
    private double[][] B={{0.0250279},{0.00210056}};
private double[][] C= {{0,18350}};
// Vector de salida, vector de estados y vector de control
private double[][] y={{0}};
    private double[][] x={{0},{0}};
private double[][] u={{0}};
    // Almacenamiento Temporal
    private double[][] Ax={{0},{0}};
    private double[][] Bu={{0},{0}};
    private ComunicacionUDP comunicacion;
    private int velocidad=0;
    public Controlador (ComunicacionUDP c) {
super(
new PriorityParameters(PriorityScheduler.instance().getMinPriority()+10)
        ,null,null,null,null,null);
comunicacion=c;
    }

    public void handleAsyncEvent() {
//Observador de Estados
        u[0][0]= comunicacion.ObtenerDato();//en RPM
        Matriz.Mult(A, 2, 2, x, 2, 1, Ax);
        Matriz.Mult(B, 2, 1, u, 1, 1, Bu);
        Matriz.Suma(Ax,2, 1,Bu,2, 1,x);
        Matriz.Mult(C, 1, 2, x, 2, 1,y);
        velocidad=(int)(0.6667*y[0][0]);
        comunicacion.EnvioDato(velocidad);
        System.out.println("u="+u[0][0]+" y="+velocidad*1.5);
comunicacion.schedulePeriodic();
    }
```

```
}

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import javax.realtime.*;
import java.io.*;
import java.net.*;
/**
 *
 * @author Félix
 */
public class RecepcionUDP extends RealtimeThread{
    private byte[] buffer_rx = new byte[1024];

    private double u=0.0;
    private final double factor = 0.0024;//supply voltaje / resolucion PWM

    public RecepcionUDP(SchedulingParameters sched,
    ReleaseParameters release){
        super(sched, release);
    }

     public void run(){
        RealtimeThread hiloactual = currentRealtimeThread();
       while(true){
            do{
                try {
                DatagramPacket datagrama_rx =
                new DatagramPacket(buffer_rx, buffer_rx.length);
                DatagramSocket socket_rx = new DatagramSocket(4406);
                socket_rx.receive(datagrama_rx);
                socket_rx.close();
                String cadena =
                new String(datagrama_rx.getData(),0,datagrama_rx.getLength());
u = factor*Integer.parseInt(cadena);//en voltios
}
                catch (IOException ioe){
                    System.out.println("IO Error: " + ioe);
}
            }while(hiloactual.waitForNextPeriod());
        }
    }

     public double ObtenerDato(){
return u;
    }

}
```

ANEXO C

Código de programa Java – GraficoJAVA

Builder.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Félix
 */
public class Builder {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

RecepcionUDP RecepcionUDP= new RecepcionUDP();
        RecepcionUDP.start();
while(true){}
    }
}
```

Dibujable.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import java.awt.Graphics;
/**
 *
 * @author Félix
 */
public interface Dibujable {

    public void setPosicion(double x, double y);
    public void dibujar(Graphics dw);

}
```

Linea.java

```
/*
 * To change this template, choose Tools | Templates
* and open the template in the editor.
*/

/**
 *
 * @author Félix
 */
public class Linea {
// definición de variables miembro de la claes
private static int numLineas = 0;
    protected double x1, y1, x2, y2;
// constructores de la clase
    public Linea(double p1x, double p1y, double p2x, double p2y) {
x1 = p1x;
    x2 = p2x;
    y1 = p1y;
    y2 = p2y;
    numLineas++;
    }

    public Linea(){ this(0, 0, 1.0, 1.0); }

}
```

LineaGrafico.java

```
/*
* To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import java.awt.Graphics;
import java.awt.Color;
/**
 *
 * @author Félix
 */
public class LineaGrafico extends Linea implements Dibujable {
    // nueva variable miembro
    Color color;
// constructor
    public LineaGrafico(double x1,double y1,double x2,double y2,Color unColor){
// llamada al constructor de Rectangulo
    super(x1, y1, x2, y2);
        this.color = unColor;     // en este caso this es opcional
    }
    // métodos de la interface Dibujable
    public void dibujar(Graphics dw) {
dw.setColor(color);
        dw.drawLine((int)x1, (int)y1, (int)(x2), (int)(y2));
}
    public void setPosicion(double x, double y) {
; // método vacío, pero necesario de definir
}
}
```

Panel Dibujo.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import java.awt.*;
import java.util.ArrayList;
import java.util.Iterator;
/**
 *
 * @author Félix
 */
public class PanelDibujo extends Panel {
// variable miembro
    private ArrayList v;
Image imgInv;
    Graphics graphInv;

// constructor
public PanelDibujo(ArrayList va) {
    super(new FlowLayout());
this.v = va;
}
// redefinición del método paint()
public void paint(Graphics g) {
    update(g);
}

public void update (Graphics g) {
    //se comprueba si existe el objeto invisible
if ((graphInv==null) ) {
        // se llama al método createImage de la clase Component
imgInv = createImage(1350,710);
        //System.out.println("hola");
// se llama al método getGraphics de la clase Image
        graphInv = imgInv.getGraphics();
    }
    // se establecen las propiedades del contexto gráfico invisible,
    //y se dibuja sobre él

    graphInv.clearRect(0, 0, 1350, 710);

    graphInv.setColor(Color.green);
    graphInv.drawLine(100,125,1300,125);
    graphInv.drawLine(100,200,1300,200);
    graphInv.drawLine(100,275,1300,275);
    graphInv.drawLine(100,350,1300,350);
    graphInv.drawLine(100,425,1300,425);
    graphInv.drawLine(100,500,1300,500);
    graphInv.drawLine(100,575,1300,575);

    graphInv.drawLine(250,50,250,650);
    graphInv.drawLine(400,50,400,650);
    graphInv.drawLine(550,50,550,650);
    graphInv.drawLine(700,50,700,650);
    graphInv.drawLine(850,50,850,650);
    graphInv.drawLine(1000,50,1000,650);
```

```
    graphInv.drawLine(1150,50,1150,650);

    graphInv.setColor(Color.black);
    graphInv.drawString (" 4000RPM", 20, 50);
    graphInv.drawString (" 3000RPM", 20, 125);
    graphInv.drawString (" 2000RPM", 20, 200);
    graphInv.drawString (" 1000RPM", 20, 275);
    graphInv.drawString ("    0RPM", 20, 350);
    graphInv.drawString ("-1000RPM", 20, 425);
    graphInv.drawString ("-2000RPM", 20, 500);
    graphInv.drawString ("-3000RPM", 20, 575);
    graphInv.drawString ("-4000RPM", 20, 650);

    graphInv.drawString ("12.s", 1300, 680);
    graphInv.drawString ("10.5s", 1150, 680);
    graphInv.drawString (" 9.0s", 1000, 680);
    graphInv.drawString (" 7.5s",  850, 680);
    graphInv.drawString (" 6.0s",  700, 680);
    graphInv.drawString (" 4.5s",  550, 680);
    graphInv.drawString (" 3.0s",  400, 680);
    graphInv.drawString (" 1.5s",  250, 680);
    graphInv.drawString (" 0.0s",  100, 680);

graphInv.drawString ("Telecontrol en Tiempo Real",  550, 40);
    //graphInv.drawString ("Telecontrol en Tiempo Real",  550, 40);
graphInv.drawRect(100, 50, 1200, 600);

    graphInv.setColor(getBackground());

    Dibujable dib;
    Iterator it;
    g.setColor(Color.red);
    it = v.iterator();
    while(it.hasNext()) {
        dib = (Dibujable)it.next();
dib.dibujar(graphInv);
    }
    // finalmente se hace visible la imagen invisible a partir del punto (0, 0)
    //utilizando el propio panel como ImageObserver
    g.drawImage(imgInv, 0, 0, this);

} // fin del método update()

} // Fin de la clase PanelDibujo
```

RecepcionUDP.java

```
/*
* To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import java.net.*;
import java.io.*;
import java.util.ArrayList;
import java.awt.*;
/**
 *
 * @author Félix
 */
public class RecepcionUDP extends Thread{
    private static byte[] buffer;
    private static LineaGrafico[] Linea = new LineaGrafico[1301];
    private DatagramSocket socket;
private DatagramPacket datagrama;
//Constructor de la clase
    public RecepcionUDP(){
}

public void run(){
            buffer = new byte [1024];
            int i=1300;
            double dato=0.0;
            double temp=0,temp1=0,temp2=300;
VentanaCerrable ventana = new VentanaCerrable("Velocidad Motor1");
ArrayList v1 = new ArrayList();
            PanelDibujo mipanel1 = new PanelDibujo(v1);
            ventana.setVisible(true);
            ventana.setSize(1350, 710);
            while(true){
                if (i==1300){
                    v1.clear();
                    i=100;
                }
                else{
                    i=i+10;
                }
                try {
                    socket = new DatagramSocket(4407);
                    datagrama = new DatagramPacket(buffer,buffer.length);
                    socket.receive(datagrama);
                    socket.close();
                    String cadena =
                    new String(datagrama.getData(),0,datagrama.getLength());
dato = Double.parseDouble(cadena);
                    System.out.println(dato);
dato = dato*0.075;// escalamiento (300/4000)RPM
temp = (int)dato;
temp1=350-(temp);
                    Linea[i] =
                    new LineaGrafico(i , temp2 , i+10,temp1, Color.black);
                    temp2=temp1;
                    v1.add(Linea[i]);
                    ventana.add(mipanel1);
```

```
             }
                catch (IOException ioe){
                    socket.close(); System.out.println("IO Error: " + ioe);
}
            }
    }
}
```

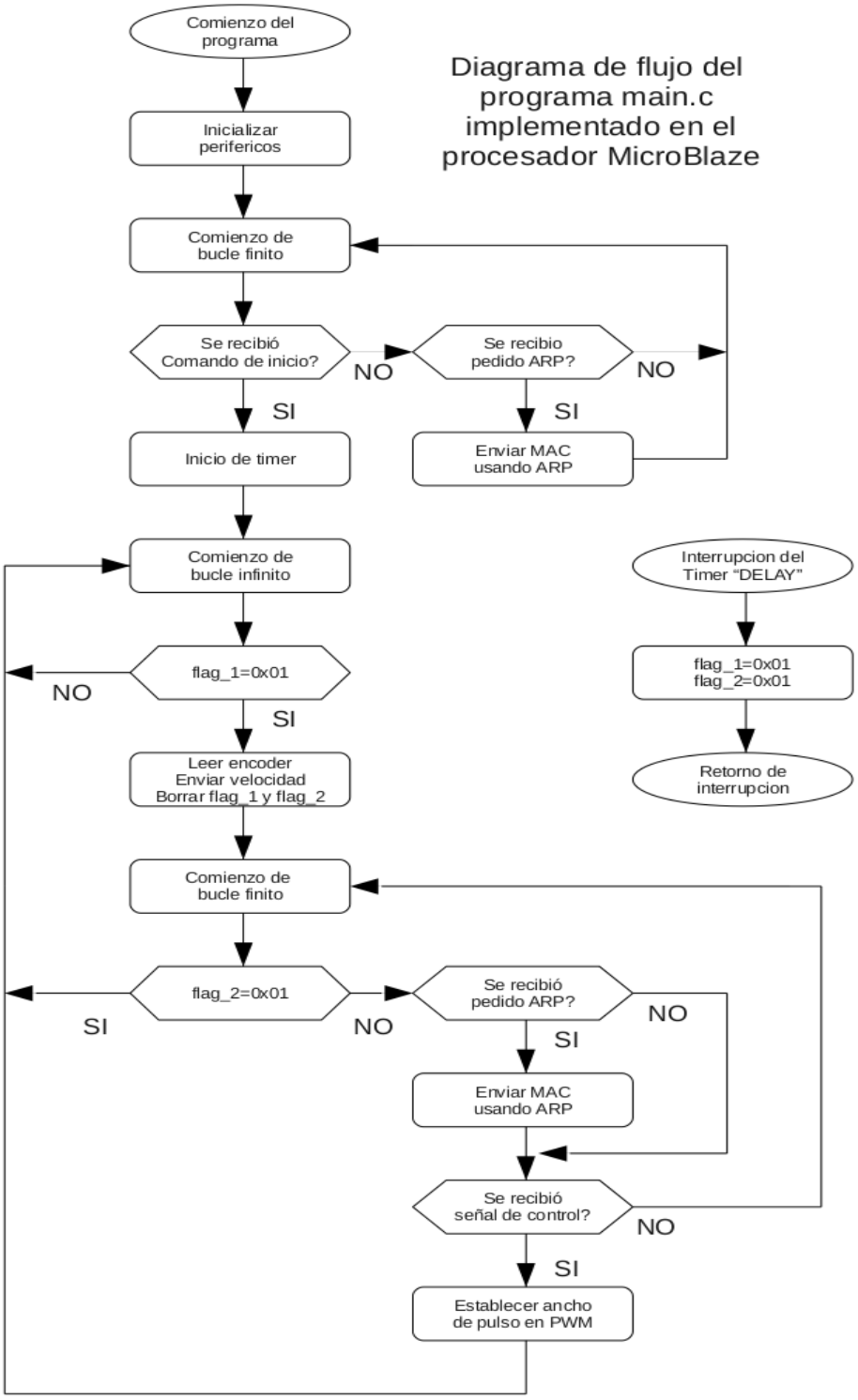
VentanaCerrable.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
*/
import java.awt.*;
import java.awt.event.*;
/**
 *
 * @author Félix
 */
class VentanaCerrable extends Frame implements WindowListener {
    // constructores
public VentanaCerrable() {
    super();
}
public VentanaCerrable(String title) {
    super(title);
    setSize(1350,720);
    addWindowListener(this);
}
// métodos de la interface WindowsListener
public void windowActivated(WindowEvent e) {;}
public void windowClosed(WindowEvent e) {;}
public void windowClosing(WindowEvent e) {System.exit(0);}
public void windowDeactivated(WindowEvent e) {;}
public void windowDeiconified(WindowEvent e) {;}
public void windowIconified(WindowEvent e) {;}
public void windowOpened(WindowEvent e) {;}

}
```

ANEXO D

Programa TelecontrolFPGA - Diagrama de flujo de Main.c



Código de programa para procesador MicroBlaze

Main.c

```
/*
    main.c -- Contiene código para aplicación de
        Telecontrol en tiempo Real
    Copyright 2011 by Universidad Nacional de Ingenieria
    License: Todos los derechos reservados
    Author: Félix Purilla <purillafloresfa@gmail.com>
Version: 2011-05-05
    Updates: no disponible
*/

#include "xparameters.h"
#include "xemaclite.h"
#include "stdio.h"
#include "string.h"
#include "xgpio_l.h"
#include "utilitario.h"
#include "interrupcion.h"
#include "eth_ip_udp.h"

volatile Xuint8 flag_1=0x00;
volatile Xuint8 flag_2=0x00;
Xuint8 instruccion=0x00;
Xuint32 Data2Pwm = 0x00000000;
// Creacion de Buffer de memoria para la transmision
//y recepcion de paquetes ethernet
volatile Xuint32 trama_Tx[LONG_MAX_TRAMA_32BITS];
volatile Xuint32 trama_Rx[LONG_MAX_TRAMA_32BITS];
volatile Xuint8 d[9]={0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30};

int main(){
//inicializacion para el manejo XEmacLite
static XEmacLite EmacLiteInstance;
   XEmacLite *EmacPtr = &EmacLiteInstance;
inicializacion_xemaclite( EmacPtr );
   // Creacion de un puntero al inicio del buffer de
   // memoria creado para la transmision.
Xuint8 *TxBuf_p;
   TxBuf_p = (Xuint8 *)trama_Tx;
// Creacion de un puntero al inicio del buffer de
   // memoria creado para la recepcion.
Xuint8 *RxBuf_p;
   RxBuf_p = (Xuint8 *)trama_Rx;
//Inicializo periferico de interrupcion
init_interrupcion(100);
int long_msje=0,long_trama=0;
   volatile unsigned long x;
   XGpio_mWriteReg(XPAR_LED_IP_0_BASEADDR,0,0x00000001);
while (instruccion!=0x06){
      instruccion=0x00;
      instruccion=XEmacLite_Recv(EmacPtr,(Xuint8 *)trama_Rx);
      if(instruccion!=0x00){
         instruccion = analisis_trama( RxBuf_p );
         if (instruccion==0x01){
         //envia MAC usando ARP
            long_trama=respuesta_arp(TxBuf_p);
```

```
XEmacLite_Send(EmacPtr,(Xuint8 *)trama_Tx,(unsigned)long_trama);

}
      }
   }
   iniciar_timer();
   //inicializacion de encoder
   XGpio_mWriteReg(XPAR_ADQ_FELIX_0_BASEADDR,0,0x00000001);
   XGpio_mWriteReg(XPAR_ADQ_FELIX_0_BASEADDR,0,0x00000000);
xil_printf("Bienvenido");
   XGpio_mWriteReg(XPAR_LED_IP_0_BASEADDR,0,0x00000002);
   while (1){
      if (flag_1==0x01){
x=XGpio_mReadReg(XPAR_ADQ_FELIX_0_BASEADDR,0);
int temp1=x>>6;
         int temp2=x>>8;
         xil_printf("Dato:%d\r\n",temp1-temp2);
         dato2string(x);
Xuint8 mesg_buf[]={d[8],d[7],d[6],d[5],d[4],d[3],d[2],d[1],d[0]};
long_msje = strlen( mesg_buf );
         long_trama = construccion_trama( mesg_buf, long_msje, TxBuf_p );

         XEmacLite_Send(EmacPtr,(Xuint8 *)trama_Tx,(unsigned)long_trama);

         flag_1=0x00;
         flag_2=0x00;
         while (instruccion!=0x05&&flag_2==0x00){
instruccion=0x00;
            instruccion=XEmacLite_Recv(EmacPtr, (Xuint8 *)trama_Rx);
            if(instruccion!=0x00){
               instruccion = analisis_trama( RxBuf_p );
               if (instruccion==0x01){
                  //envia MAC usando ARP
                  long_trama=respuesta_arp(TxBuf_p);
XEmacLite_Send(EmacPtr,(Xuint8 *)trama_Tx,(unsigned)long_trama);
               }
               if (instruccion==0x05){
                  //establece PWM
                  XGpio_mWriteReg(XPAR_PWM_FELIX_0_BASEADDR,0,Data2Pwm);
                  XGpio_mWriteReg(XPAR_LED_IP_0_BASEADDR,0,0x00000004);
}
            }
         }
      }
   }
   return 0;
}
```

Estructuras.h

```
/*
    estructuras.h -- Contiene las estructuras de las
 cabeceras ethernet, ip, udp y arp, contiene también las
 direcciones MAC, IP y UDP destino y origen.
    Copyright 2011 by Universidad Nacional de Ingeniería
    License: Todos los derechos reservados
    Author: Félix Purilla <purillafloresfa@gmail.com>
Version: 2011-05-05
    Updates: no disponible
*/

#ifndef ESTRUCTURAS_H           /* prevent circular inclusions */
#define ESTRUCTURAS_H           /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

#include "xparameters.h"
//*********************************************************************
//MAC , IP y Puerto UDP origen, esto es del FPGA
Xuint8 org_mac_addr[] = { 0x00, 0x18, 0x3e, 0x00, 0x01, 0x09 };
Xuint8 org_ip_addr[] = { 200, 37, 46, 180 };
Xuint16 org_puerto = 0x1136;//4406
//MAC , IP y Puerto UDP destino, esto es de la PC
Xuint8 dst_mac_addr[] = { 0x00, 0x1e, 0x33, 0xfb, 0xb7, 0x25 };
//Xuint8 dst_mac_addr[] = { 0x00, 0x05, 0x5d, 0x8c, 0xee, 0x6c };
//Xuint8 dst_mac_addr[] = { 0x70, 0x71, 0xbc, 0x09, 0x21, 0x45 };
Xuint8 dst_ip_addr[] = { 200, 37, 46, 189 };
Xuint16 dst_puerto = 0x1135;//4405
//*********************************************************************
//Creacion de un struct para la cabecera ethernet
struct trama_eth{
   Xuint8 mac_dst[6];
   Xuint8 mac_org[6];
Xuint16 type_length;
};
//Creacion de un struct para la cabecera ip
struct trama_ip {
   Xuint16 v_hl_tos;
   Xuint16 len;
   Xuint16 id;
   Xuint16 offset;
   Xuint16 ttl_proto;
   Xuint16 chksum;
   Xuint8 ip_org[4];
   Xuint8 ip_dst[4];
};
//Creacion de un struct para la cabecera udp
struct trama_udp {
   Xuint16 udp_org;
Xuint16 udp_dst;
   Xuint16 udp_len;
Xuint16 udp_chksum;
};
//Creacion de un struct para la respuesta ARP
```

```
struct trama_arp {
   Xuint16 hw_type;
   Xuint16 prot_type;
   Xuint8 hw_size;
Xuint8 prot_size;
   Xuint16 opcode;
   Xuint8 mac_org[6];
   Xuint8 ip_org[4];
   Xuint8 mac_dst[6];
Xuint8 ip_dst[4];
};

#define LONG_MAX_TRAMA_8BITS XEL_MAX_FRAME_SIZE
#define LONG_MAX_TRAMA_32BITS ((LONG_MAX_TRAMA_8BITS / sizeof(Xuint32)) + 1)
#define SIZE_ETHERNET_HEADER 14
#define SIZE_IP_HEADER 20
#define SIZE_UDP_HEADER 8
#define SIZE_ARP_BODY 28

#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

Eth_ip_udp.h

```
/*
    eth_ip_udp.h -- Contiene las funciones necesarias para construir y
    analizar tramas correspondientes a protocolo UDP, así también posee
    una función para inicializar el modulo XEmacLite y enviar respuestas
    en protocolo ARP.
    Copyright 2011 by Universidad Nacional de Ingeniería
    License: Todos los derechos reservados
    Author: Félix Purilla <purillafloresfa@gmail.com>
Version: 2011-05-05
    Updates: no disponible
*/

#ifndef ETH_IP_UDP_H            /* prevent circular inclusions */
#define ETH_IP_UDP_H            /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

#include "estructuras.h"
#include "utilitario.h"
#include "xparameters.h"//contiene los parametros y direcciones de memoria de los
driver

extern Xuint32 Data2Pwm;

//*************************************************************
void inicializacion_xemaclite( XEmacLite *EmacPtr ) {
XEmacLite_Initialize(EmacPtr, XPAR_ETHERNET_MAC_DEVICE_ID);
   XEmacLite_SetMacAddress(EmacPtr, org_mac_addr);
XEmacLite_FlushReceive(EmacPtr);
}
//Funcion para el envio de MAC usando ARP
Xuint16 respuesta_arp( Xuint8 *buf ) {
   Xuint8 *buf_p;
//Establecimiento de la trama ARP
buf_p = buf + SIZE_ETHERNET_HEADER ;
struct trama_arp *aTr_p;
   aTr_p = (struct trama_arp *)buf_p ;
   aTr_p->hw_type   = 0x0001;
   aTr_p->prot_type = 0x0800;
   aTr_p->hw_size   = 0x06;
   aTr_p->prot_size = 0x04;
   aTr_p->opcode    = 0x0002;
   copiar_datos( org_mac_addr, aTr_p->mac_org, 6 );
copiar_datos( org_ip_addr , aTr_p->ip_org , 4 );
copiar_datos( dst_mac_addr, aTr_p->mac_dst, 6 );
   copiar_datos( dst_ip_addr , aTr_p->ip_dst , 4 );
//Establecimiento de la cabecera Ethernet
buf_p = buf ;
struct trama_eth *eTr_p;
   eTr_p = (struct trama_eth *)buf_p;
   eTr_p->type_length = 0x0806;
   copiar_datos( dst_mac_addr, eTr_p->mac_dst, 6 );
   copiar_datos( org_mac_addr, eTr_p->mac_org, 6 );
   return SIZE_ETHERNET_HEADER+SIZE_ARP_BODY;
```

```
}
//*************************************************************
int construccion_trama( Xuint8 *buf, int abuf_len, Xuint8 *rpybuf ){
//creacion de puntero a la trama
Xuint8 *rbuf_p;
   int chksum2, reply_len;
   rbuf_p = rpybuf+SIZE_ETHERNET_HEADER+SIZE_IP_HEADER+SIZE_UDP_HEADER;
copiar_datos( buf, rbuf_p, abuf_len );
   reply_len = abuf_len;
   //Establecimiento de la cabecera UDP
struct trama_udp *uTr_p;
   rbuf_p = rpybuf + SIZE_ETHERNET_HEADER + SIZE_IP_HEADER ;
   uTr_p = (struct trama_udp *)rbuf_p;
uTr_p->udp_org = org_puerto;
uTr_p->udp_dst = dst_puerto;
   uTr_p->udp_len = reply_len + SIZE_UDP_HEADER;
   uTr_p->udp_chksum = 0;
   reply_len += SIZE_UDP_HEADER;
//Establecimiento de la cabecera IP
struct trama_ip *iTr_p;
rbuf_p = rpybuf + SIZE_ETHERNET_HEADER ;
   iTr_p = (struct trama_ip *)rbuf_p ;
   iTr_p->v_hl_tos = 0x4500;
   iTr_p->len =reply_len + SIZE_IP_HEADER;
   iTr_p->id = 0x0000;
   iTr_p->offset = 0x0000;
   iTr_p->ttl_proto = 0x1111;
   iTr_p->chksum = 0;
   chksum2 = chksum16( rbuf_p, SIZE_IP_HEADER, 0 );
   chksum2 = (~chksum2) & 0xffff;
   iTr_p->chksum = chksum2;
   reply_len += SIZE_IP_HEADER;
   copiar_datos( org_ip_addr, iTr_p->ip_org, 4 );
   copiar_datos( dst_ip_addr, iTr_p->ip_dst, 4 );
//Establecimiento de la cabecera Ethernet
struct trama_eth *eTr_p;
rbuf_p = rpybuf ;
   eTr_p = (struct trama_eth *)rbuf_p;
   eTr_p->type_length = 0x0800;
   copiar_datos( dst_mac_addr, eTr_p->mac_dst, 6 );
copiar_datos( org_mac_addr, eTr_p->mac_org, 6 );
   reply_len += SIZE_ETHERNET_HEADER;
return reply_len;
}
//*************************************************************
//Funcion para analizar y procesar la trama recibida
Xuint8 analisis_trama( Xuint8 *buf) {
   Xuint32 temp          = 0x00000000;
   Xuint32 sentido       = 0x00000000;
int length_trama,i;
   Xuint8 protocoloUDP;
   Xuint16 ttl_proto;
   struct trama_eth *eTr_p;
   eTr_p = (struct trama_eth *)buf;
   if ( eTr_p->type_length == 0x0806 ){
      return 0x01;
   }
   if ( eTr_p->type_length != 0x0800 ){
```

```
      return 0x02;
   }
   if (
      eTr_p->mac_dst[0]!=org_mac_addr[0]||
      eTr_p->mac_dst[1]!=org_mac_addr[1]||
      eTr_p->mac_dst[2]!=org_mac_addr[2]||
      eTr_p->mac_dst[3]!=org_mac_addr[3]||
      eTr_p->mac_dst[4]!=org_mac_addr[4]||
      eTr_p->mac_dst[5]!=org_mac_addr[5]){
      return 0x03;
   }
   buf += SIZE_ETHERNET_HEADER;
   struct trama_ip *iTr_p;
   iTr_p = (struct trama_ip *) buf;
ttl_proto=iTr_p->ttl_proto;
protocoloUDP=ttl_proto;
   if ( protocoloUDP != 0x11 ){
      return 0x04;
   }
   buf +=SIZE_IP_HEADER;
   struct trama_udp *uTr_p;
   uTr_p = (struct trama_udp *) buf;
   length_trama=uTr_p->udp_len - SIZE_UDP_HEADER;
buf += SIZE_UDP_HEADER;
   if (buf[0]==0x41) return 0x06;
   for(i=0; i<length_trama ; i++) {
      switch(buf[i]){
         case 0x2D:sentido=0x0001;break;
         case 0x30:temp=temp*10+0;break;
         case 0x31:temp=temp*10+1;break;
         case 0x32:temp=temp*10+2;break;
         case 0x33:temp=temp*10+3;break;
         case 0x34:temp=temp*10+4;break;
         case 0x35:temp=temp*10+5;break;
         case 0x36:temp=temp*10+6;break;
         case 0x37:temp=temp*10+7;break;
         case 0x38:temp=temp*10+8;break;
         case 0x39:temp=temp*10+9;break;
         defaul:break;
      }
   }
   Data2Pwm=((sentido<<16)|temp);
return 0x05;
}

#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

Interrupcion.h

```
/*
    interrupcion.h -- Contiene tres funciones, la primera es la función
    que será llamada cada vez que se produzca una interrupción, la segunda
    es una función que inicializa el modulo XPS_INTC y XPS_TIMER y la
    tercera función inicializa la cuenta del timer y habilita las interrupciones
    en el procesador MicroBlaze.
    Copyright 2011 by Universidad Nacional de Ingeniería
    License: Todos los derechos reservados
    Author: Félix Purilla <purillafloresfa@gmail.com>
Version: 2011-05-05
    Updates: no disponible
*/

#ifndef INTERRUPCION_H          /* prevent circular inclusions */
#define INTERRUPCION_H          /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif

#include "xparameters.h"
#include "xtmrctr.h"
#include "xintc_l.h"

extern volatile Xuint8 flag_1;
extern volatile Xuint8 flag_2;

//funcion para manejar la interrupcion
void manejador_interrupcion(void * baseaddr_p){
   Xuint32 csr;
   //En csr almaceno el registro de estados
csr = XTmrCtr_mGetControlStatusReg(XPAR_DELAY_BASEADDR, 0);
   flag_1=0x01;
   flag_2=0x01;
   XTmrCtr_mSetControlStatusReg(XPAR_DELAY_BASEADDR, 0, csr);
}

void init_interrupcion(int x) {//x en milisegundos
XIntc_RegisterHandler(XPAR_BREAK_BASEADDR,XPAR_BREAK_DELAY_INTERRUPT_INTR,
(XInterruptHandler) manejador_interrupcion,(void *)XPAR_DELAY_BASEADDR);
   //Habilita la generacion de la interrupcion IRQ
XIntc_mMasterEnable(XPAR_BREAK_BASEADDR);
//Habilita la respuesta al pedido de interrupcion de la instancia break
XIntc_mEnableIntr(XPAR_BREAK_BASEADDR, XPAR_DELAY_INTERRUPT_MASK);
//Tiempo entre interrupciones
XTmrCtr_mSetLoadReg(XPAR_DELAY_BASEADDR, 0, x* 50000);
//reset del XPS TIMER
}

void iniciar_timer(void){
XTmrCtr_mSetControlStatusReg(XPAR_DELAY_BASEADDR, 0, XTC_CSR_INT_OCCURED_MASK |
XTC_CSR_LOAD_MASK );
   //inicio del TIMER
   XTmrCtr_mSetControlStatusReg(XPAR_DELAY_BASEADDR, 0, XTC_CSR_ENABLE_TMR_MASK |
   XTC_CSR_ENABLE_INT_MASK | XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK);
```

```
//Habilita la respuesta a los pedidos de interrupcion en el MicroBlaze
microblaze_enable_interrupts();
}

#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

Utilitario.h

```
/*
    utilitario.h -- Contiene tres funciones, la primera hace
    copia de datos de un espacio de memoria a otro, la segunda función
    recibe un numero de 32 bits y almacena en un espacio memoria
    su equivalente en código ASCII, y la tercera función obtiene la
    suma de cabecera según norma RFC791.
    Copyright 2011 by Universidad Nacional de Ingeniería
    License: Todos los derechos reservados
    Author: Félix Purilla <purillafloresfa@gmail.com>
Version: 2011-05-05
    Updates: no disponible
*/
#ifndef UTILITARIO_H            /* prevent circular inclusions */
#define UTILITARIO_H            /* by using protection macros */

#ifdef __cplusplus
extern "C" {
#endif


extern volatile Xuint8 d[9];

void copiar_datos( void *org, void *dst, int n ) {
   Xuint8 *org_p = (Xuint8 *)org;
Xuint8 *dst_p = (Xuint8 *)dst;
while(n--) {
   dst_p[n] = org_p[n];
   }
}

void dato2string( unsigned long x) {
   unsigned long temp,resto;
   Xuint8 giro;
   int i;
   giro=x;
   temp = x>>8;
   x=temp;
   for(i=0;i<8;i++){
      resto=temp%10;
      temp=temp/10;
      switch(resto){
         case 0:d[i]=0x30;break;
         case 1:d[i]=0x31;break;
         case 2:d[i]=0x32;break;
         case 3:d[i]=0x33;break;
         case 4:d[i]=0x34;break;
```

```
         case 5:d[i]=0x35;break;
         case 6:d[i]=0x36;break;
         case 7:d[i]=0x37;break;
         case 8:d[i]=0x38;break;
         case 9:d[i]=0x39;break;
         default:d[i]=0x41;
      }
   }
   if (giro==2)
      d[8]=0x2B;
   else
      d[8]=0x2D;
}

int chksum16( void *buf1, short len, int chksum ) {
   unsigned short *buf = buf1;
   int chksum16;
   while(len>0) {
      if (len==1)
         chksum16 = ((*buf) & 0xFF00 );
      else
         chksum16 = (*buf);
      chksum += chksum16;
      *buf++;
      len -= 2;
   }
   while( chksum >> 16 )
      chksum = (chksum & 0x0FFFF) + (chksum >> 16);
return chksum;
}

#ifdef __cplusplus
}
#endif

#endif /* end of protection macro */
```

ANEXO E

Código de la IP core del modulador por ancho de pulso

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ****************************************************************************
-- ** Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.            **
-- **                                                                        **
-- ** Xilinx, Inc.                                                           **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND        **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,         **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,         **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION            **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,      **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        **
-- ** FOR A PARTICULAR PURPOSE.                                              **
-- **                                                                        **
-- ****************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Tue Nov 23 11:12:29 2010 (by Create and Import Peripheral
Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
```

```
--   component instantiations:             "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------
-- user_logic.vhd -- Implementa modulador por ancho de pulso cuyo ciclo de
-- trabajo puede ser establecido a peticion del procesador MicroBlaze
-- Copyright 2011 by Universidad Nacional de Ingenieria
-- License: Todos los derechos reservados
-- Author: Félix Purilla <purillafloresfa@gmail.com>
-- Version: 2011-05-05
-- Updates: no disponible
------------------------------------------------------------------------------
-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                -- Slave interface data bus width
--   C_NUM_REG                   -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                  -- Bus to IP clock
--   Bus2IP_Reset                -- Bus to IP reset
--   Bus2IP_Data                 -- Bus to IP data bus
--   Bus2IP_BE                   -- Bus to IP byte enables
--   Bus2IP_RdCE                 -- Bus to IP read chip enable
--   Bus2IP_WrCE                 -- Bus to IP write chip enable
--   IP2Bus_Data                 -- IP to Bus data bus
--   IP2Bus_RdAck                -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                -- IP to Bus error response
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
       --resolucion pwm=rPWM=50*20ns=1us
      N                             :INTEGER              :=8;
       M                             :INTEGER              :=50;
       frec                          :INTEGER              :=10000;
-- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH                  : integer             := 32;
    C_NUM_REG                     : integer             := 1
```

```
    -- DO NOT EDIT ABOVE THIS LINE --------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
       PWM_CTRL                      : out std_logic;
       PWM_X                         : out std_logic;
       PWM_Y                         : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------
    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Reset                   : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
  signal PWM_CTRL_s                       : std_logic;
  signal PWM_X_s                          : std_logic;
  signal PWM_Y_s                          : std_logic;
  signal giro_actual,giro_sgte            : std_logic;
  ----------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ----------------------------------------
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;
  --======instanciacion componente base_tiempo===========
  signal tick_out:STD_LOGIC;
  signal r_actual,r_sgte:unsigned(0 to N-1);
  --==== ==================
  signal reiniciar:STD_LOGIC;
--==Registro que almacena el valor del contador
signal cont_actual,cont_sgte:unsigned(0 to 15);
--==Registro que almacena el valor de la duracion del pulso PWM
signal tau_actual,tau_sgte:unsigned(0 to 15);
--==Flip Flop que permite registrar la salida PWM
signal pwm_actual,pwm_sgte:STD_LOGIC;
```

```
  --------------------------------------------
begin
--USER logic implementation added here

--==========base de tiempo para establecer la resolucion PWM==============
process(Bus2IP_Clk,Bus2IP_Reset)
begin
       if Bus2IP_Reset='1' then
              r_actual<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
              r_actual<=r_sgte;
       end if;
end process;
r_sgte    <=  (others=>'0') when r_actual=M-1 else
                          r_actual+1;

tick_out  <=  '1' when r_actual=M-1 else
                    '0';--genera un pulso cuando se llega a M-1

--======================Registro Contador====================
process(Bus2IP_Clk,Bus2IP_Reset)
begin
       if Bus2IP_Reset='1' then
              cont_actual<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
              cont_actual<=cont_sgte;
       end if;
end process;
cont_sgte<= (others=>'0') when reiniciar='1' else
                           cont_actual+1 when tick_out='1'  else
                           cont_actual;

--==============Registro de estados==========================
process(Bus2IP_Clk,Bus2IP_Reset)
begin
       if Bus2IP_Reset='1' then
              tau_actual<=(others=>'0');
              pwm_actual<='0';
              giro_actual<='0';
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
              tau_actual<=tau_sgte;
              pwm_actual<=pwm_sgte;
              giro_actual<=giro_sgte;
       end if;
end process;

tau_sgte<= unsigned(Bus2IP_Data(16 to 31)) when Bus2IP_WrCE(0) = '1' else
                      tau_actual;

giro_sgte<=Bus2IP_Data(15) when Bus2IP_WrCE(0) = '1' else
                      giro_actual;

reiniciar<= '1' when cont_actual=frec else
                           '0';

pwm_sgte<=  '1' when cont_actual<tau_actual else
                           '0';
```

```
PWM_CTRL_s<=pwm_actual;
PWM_X_s<=PWM_CTRL_s and (not (giro_actual));
PWM_Y_s<=PWM_CTRL_s and giro_actual;

PWM_X    <= not PWM_X_s;
PWM_Y    <= not PWM_Y_s;
PWM_CTRL <= not '1';

slv_write_ack     <= Bus2IP_WrCE(0);
slv_read_ack      <= Bus2IP_RdCE(0);
IP2Bus_WrAck <= slv_write_ack;
IP2Bus_RdAck <= slv_read_ack;
IP2Bus_Error <= '0';

end IMP;
```

Resumen del informe de la síntesis de la IP core PWM

```
=========================================================================
HDL Synthesis Report
=========================================================================
Macro Statistics
# Adders/Subtractors                                   : 2
 13-bit adder                                          : 1
 9-bit subtractor                                      : 1
# Registers                                            : 40
 1-bit register                                        : 28
 13-bit register                                       : 1
 16-bit register                                       : 2
 3-bit register                                        : 1
 32-bit register                                       : 4
 4-bit register                                        : 3
 9-bit register                                        : 1
# Comparators                                          : 1
 16-bit comparator less                                : 1
=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================
Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <pwm_felix_0/USER_LOGIC_I/estado_actual/FSM> on signal
<estado_actual[1:2]> with gray encoding.
-------------------
 State | Encoding
-------------------
 s0    | 00
 s1    | 01
 s2    | 11
 s3    | 10
-------------------
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-4
 Number of Slices:                       75  out of   4656     1%
 Number of Slice Flip Flops:            101  out of   9312     1%
 Number of 4 input LUTs:                 75  out of   9312     0%
 Number of IOs:                         204
 Number of bonded IOBs:                   0  out of    232     0%
---------------------------
Timing Summary:
---------------
Speed Grade: -4
   Minimum period: 5.421ns (Maximum Frequency: 184.468MHz)
   Minimum input arrival time before clock: 2.377ns
   Maximum output required time after clock: 1.917ns
   Maximum combinational path delay: No path found
```

ANEXO F

Código de la IP core de adquisición de velocidad

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--
-- ****************************************************************************
-- ** Copyright (c) 1995-2008 Xilinx, Inc.  All rights reserved.            **
-- **                                                                        **
-- ** Xilinx, Inc.                                                           **
-- ** XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS"          **
-- ** AS A COURTESY TO YOU, SOLELY FOR USE IN DEVELOPING PROGRAMS AND        **
-- ** SOLUTIONS FOR XILINX DEVICES.  BY PROVIDING THIS DESIGN, CODE,         **
-- ** OR INFORMATION AS ONE POSSIBLE IMPLEMENTATION OF THIS FEATURE,         **
-- ** APPLICATION OR STANDARD, XILINX IS MAKING NO REPRESENTATION            **
-- ** THAT THIS IMPLEMENTATION IS FREE FROM ANY CLAIMS OF INFRINGEMENT,      **
-- ** AND YOU ARE RESPONSIBLE FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE       **
-- ** FOR YOUR IMPLEMENTATION.  XILINX EXPRESSLY DISCLAIMS ANY               **
-- ** WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE                **
-- ** IMPLEMENTATION, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR         **
-- ** REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF        **
-- ** INFRINGEMENT, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS        **
-- ** FOR A PARTICULAR PURPOSE.                                              **
-- **                                                                        **
-- ****************************************************************************
--
------------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Tue Oct 05 23:19:12 2010 (by Create and Import Peripheral
Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
```

```
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------
-- user_logic.vhd -- Implementa un sistema digital que obtiene la velocidad
-- del eje de un motor mediante el procesamiento de dos señales en cuadratura
-- proveniente de un encoder.
-- Copyright 2011 by Universidad Nacional de Ingenieria
-- License: Todos los derechos reservados
-- Author: Félix Purilla <purillafloresfa@gmail.com>
-- Version: 2011-05-05
-- Updates: no disponible
------------------------------------------------------------------------------
-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_DWIDTH                 -- Slave interface data bus width
--   C_NUM_REG                    -- Number of software accessible registers
--
-- Definition of Ports:
--   Bus2IP_Clk                   -- Bus to IP clock
--   Bus2IP_Reset                 -- Bus to IP reset
--   Bus2IP_Data                  -- Bus to IP data bus
--   Bus2IP_BE                    -- Bus to IP byte enables
--   Bus2IP_RdCE                  -- Bus to IP read chip enable
--   Bus2IP_WrCE                  -- Bus to IP write chip enable
--   IP2Bus_Data                  -- IP to Bus data bus
--   IP2Bus_RdAck                 -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                 -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                 -- IP to Bus error response
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
        --Base de tiempo de 800us: 800us/20ns=40000
       N_bt1                  :INTEGER         :=22;
        M_bt1                  :INTEGER         :=2250000;--2250000*20ns=45ms
        --Base de tiempo de 800us: 800us/20ns=40000
        N_bt2                  :INTEGER         :=22;
        M_bt2                  :INTEGER         :=2500000;--2500000*20ns=50ms
        --Base de tiempo de 800us: 800us/20ns=40000
        N_bt3                  :INTEGER         :=18;
        M_bt3                  :INTEGER         :=250000;--250000*20ns=5ms
```

```
        --Notese que: M_bt1+M_bt2+M_bt3=tiempo_de_muestreo/20ns
        --Tamaño del registro de cuenta de flancos de subida
        N_cf                    :INTEGER         :=15;
        --Base de tiempo para debouncing
        N_db                    :INTEGER         :=15;
        M_db                    :INTEGER         :=900;--optimo a 900
-- ADD USER GENERICS ABOVE THIS LINE ---------------
    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_DWIDTH           : integer        := 32;
    C_NUM_REG              : integer        := 1
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
        ADQ                             : in std_logic_vector(0 to 1);
    -- ADD USER PORTS ABOVE THIS LINE ------------------
    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Reset                   : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
-- Señales para el bloque comunicacion
  signal rx_actual,rx_sgte,rx,tx        :std_logic_vector(0 to 31);
signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;
signal inicio                         : std_logic;
  -- Señales para base de tiempo 1
type estado_bt1 is(s0,s1,s2);
signal estado_actual_bt1,estado_sgte_bt1: estado_bt1;
signal cont_actual_bt1,cont_sgte_bt1  :std_logic_vector(0 to N_bt1);
  signal tick_bt1                       : std_logic;
-- Señales para base de tiempo 2
```

```
type estado_bt2 is(s0,s1,s2);
signal estado_actual_bt2,estado_sgte_bt2: estado_bt2;
signal cont_actual_bt2,cont_sgte_bt2    :std_logic_vector(0 to N_bt2);
  signal tick_bt2                  : std_logic;
-- Señales para base de tiempo 3
type estado_bt3 is(s0,s1,s2);
signal estado_actual_bt3,estado_sgte_bt3: estado_bt3;
signal cont_actual_bt3,cont_sgte_bt3    :std_logic_vector(0 to N_bt3);
  signal tick_bt3                  : std_logic;
--==== Bloque base de tiempo para debouncing db  =======
type   estado_db is(s0,s1,s2);
signal estado_actual_db,estado_sgte_db: estado_db          ;
signal cont_actual_db,cont_sgte_db    : unsigned(0 to N_db);
  signal tick_db                     : std_logic          ;
--==== Bloque detector de flancos de subida df    =======
type   estado_df is(s0,s1,s2,s3,s4,s5);
signal estado_actual_df,estado_sgte_df: estado_df          ;
signal tick_df                        : std_logic          ;
  signal flag_df                      : std_logic          ;
--==== Bloque contador de flancos de subida cf  =======
signal cont_actual_cf,cont_sgte_cf    : unsigned(0 to N_cf);
  signal pps_actual_cf,pps_sgte_cf    : std_logic_vector(0 to N_cf);
  signal pps_cf                       : std_logic_vector(0 to N_cf);
-----
  --==== Bloque envio de datos al procesador       =======
  signal giro,giro_sgte,giro_actual     : std_logic_vector(0 to 7);
  --==== Bloque deteccion de flancos de subida en ADQ(1)==
type estado_B is(s0,s1);
signal estado_actual_B,estado_sgte_B : estado_B;
signal s_tick_B:std_logic;
--==== Bloque obtencion de sentido de giro       ========
signal try1,try2,try3:std_logic;
begin

  --************************Inicio Bloque*********************************
  --====Implementacion de la base de tiempo 1                        =======
  --====Las senales con terminacion _bt1 implican la pertenenencia   =======
  --====a este bloque                                                =======
  --====Senales de entrada: Bus2IP_Reset,Bus2IP_Clk,tick_bt3,inicio  =======
  --====Senales internas  : cont_actual_db,cont_sgte_bt1             =======
  --====                  : estado_actual_bt1,estado_sgte_bt1        =======
--====Senales de salida : tick_bt1  =========
process(Bus2IP_Clk,Bus2IP_Reset)
  begin
    if Bus2IP_Reset='1' then
          estado_actual_bt1<=s0;
      cont_actual_bt1<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
            estado_actual_bt1<=estado_sgte_bt1;
            cont_actual_bt1<=cont_sgte_bt1;
       end if;
  end process;

  process(estado_actual_bt1,cont_actual_bt1,inicio,tick_bt3)
begin
    estado_sgte_bt1<=estado_actual_bt1;
       cont_sgte_bt1<=cont_actual_bt1;
       tick_bt1<='0';
```

```
        case estado_actual_bt1 is
        when s0=>
              if inicio='1' or tick_bt3='1'then
                    estado_sgte_bt1<=s1;
              end if;
        when s1=>
              if cont_actual_bt1=M_bt1 then
                    estado_sgte_bt1<=s2;
                    cont_sgte_bt1<=(others=>'0');
              else
                    cont_sgte_bt1<=cont_actual_bt1+1;
              end if;
        when s2=>
              tick_bt1<='1';
              estado_sgte_bt1<=s0;
        end case;
end process;
  --***************************Fin Bloque*****************************

  --************************Inicio Bloque*****************************
  --====Implementacion de la base de tiempo 2                    =======
  --====Las senales con terminacion _bt2 implican la pertenenencia=======
  --====a este bloque                                            =======
  --====Senales de entrada: Bus2IP_Reset,Bus2IP_Clk,tick_bt1     =======
  --====Senales internas  : cont_actual_bt2,cont_sgte_bt2        =======
  --====                  : estado_actual_bt2,estado_sgte_bt2    =======
--====Senales de salida : tick_bt2                              =======
process(Bus2IP_Clk,Bus2IP_Reset)
  begin
    if Bus2IP_Reset='1' then
         estado_actual_bt2<=s0;
      cont_actual_bt2<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
             estado_actual_bt2<=estado_sgte_bt2;
             cont_actual_bt2<=cont_sgte_bt2;
       end if;
  end process;

  process(estado_actual_bt2,cont_actual_bt2,tick_bt1)
begin
    estado_sgte_bt2<=estado_actual_bt2;
       cont_sgte_bt2<=cont_actual_bt2;
       tick_bt2<='0';
       case estado_actual_bt2 is
       when s0=>
             if tick_bt1='1'then
                    estado_sgte_bt2<=s1;
             end if;
       when s1=>
             if cont_actual_bt2=M_bt2 then
                    estado_sgte_bt2<=s2;
                    cont_sgte_bt2<=(others=>'0');
             else
                    cont_sgte_bt2<=cont_actual_bt2+1;
             end if;
       when s2=>
             tick_bt2<='1';
             estado_sgte_bt2<=s0;
```

```
        end case;
end process;
  --****************************Fin Bloque*********************************

  --************************Inicio Bloque*********************************
  --====Implementacion de la base de tiempo de 500*20ns=10us para  =======
  --====evitar contar como flancos validos el rebote en encoder    =======
  --====Las senales con terminacion _bt3 implican la pertenenencia =======
  --====a este bloque                                             =======
  --====Senales de entrada: Bus2IP_Reset,Bus2IP_Clk,tick_bt2       =======
  --====Senales internas  : cont_actual_bt3,cont_sgte_bt3          =======
  --====                  : estado_actual_bt2,estado_sgte_bt2      =======
--====Senales de salida : tick_bt3                                 =======
process(Bus2IP_Clk,Bus2IP_Reset)
  begin
    if Bus2IP_Reset='1' then
          estado_actual_bt3<=s0;
      cont_actual_bt3<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
            estado_actual_bt3<=estado_sgte_bt3;
            cont_actual_bt3<=cont_sgte_bt3;
       end if;
  end process;

  process(estado_actual_bt3,cont_actual_bt3,tick_bt2)
begin
    estado_sgte_bt3<=estado_actual_bt3;
       cont_sgte_bt3<=cont_actual_bt3;
        tick_bt3<='0';
        case estado_actual_bt3 is
        when s0=>
             if tick_bt2='1'then
                   estado_sgte_bt3<=s1;
             end if;
        when s1=>
             if cont_actual_bt3=M_bt3 then
                   estado_sgte_bt3<=s2;
                   cont_sgte_bt3<=(others=>'0');
             else
                   cont_sgte_bt3<=cont_actual_bt3+1;
             end if;
        when s2=>
             tick_bt3<='1';
             estado_sgte_bt3<=s0;
        end case;
end process;
  --****************************Fin Bloque*********************************

  --************************Inicio Bloque*********************************
  --====Implementacion de la base de tiempo de 500*20ns=10us para     =========
  --====evitar contar como flancos validos el rebote en encoder       =========
  --====Las senales con terminacion _db implican la pertenenencia     =========
  --====a este bloque                                               =========
  --====Senales de entrada: Bus2IP_Reset,Bus2IP_Clk,flag_df          =========
  --====Senales internas  : cont_actual_db,cont_sgte_db              =========
  --====                  : estado_actual_db,estado_sgte_db          =========
--====Senales de salida : tick_db                                   =========
process(Bus2IP_Clk,Bus2IP_Reset)
```

```
  begin
    if Bus2IP_Reset='1' then
          estado_actual_db<=s0;
      cont_actual_db<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
            estado_actual_db<=estado_sgte_db;
            cont_actual_db<=cont_sgte_db;
       end if;
  end process;

  process(estado_actual_db,cont_actual_db)
begin
    estado_sgte_db<=estado_actual_db;
       cont_sgte_db<=cont_actual_db;
        tick_db<='0';
        case estado_actual_db is
        when s0=>
             if flag_df='1' then
                   estado_sgte_db<=s1;
             end if;
        when s1=>
             if cont_actual_db=M_db then
                   estado_sgte_db<=s2;
                   cont_sgte_db<=(others=>'0');
             else
                   cont_sgte_db<=cont_actual_db+1;
             end if;
        when s2=>
             tick_db<='1';
             estado_sgte_db<=s0;
        end case;
end process;
  --****************************Fin Bloque*************************************


  --************************Inicio Bloque*************************************
  --====Implementacion de una FSM que detecta flancos de subida       =========
  --====                                                             =========
  --====Las senales con terminacion _df implican la pertenenencia     =========
  --====a este bloque                                                =========
  --====Senales de entrada: Bus2IP_Reset,Bus2IP_Clk,ADQ(0),tick_db    =========
  --====Senales internas  : estado_actual_df,estado_sgte_df           =========
--====                  :                                          =========
  --====Senales de salida : tick_df,flag_df                          =========
process(Bus2IP_Clk,Bus2IP_Reset)
  begin
    if Bus2IP_Reset='1' then
      estado_actual_df<=s0;
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
             estado_actual_df<=estado_sgte_df;
       end if;
  end process;

  process(estado_actual_df,ADQ(0),tick_db)
begin
       estado_sgte_df<=estado_actual_df;
       tick_df<='0';
       flag_df<='0';
```

```
      case estado_actual_df is
      when s0=>
            if ADQ(0)='1' then
                  estado_sgte_df<=s1;
            end if;
      when s1=>
            tick_df<='1';
            flag_df<='1';
            estado_sgte_df<=s2;
      when s2=>
            if tick_db='1' then
                  estado_sgte_df<=s3;
            end if;
      when s3=>
            if ADQ(0)='0' then
                  estado_sgte_df<=s4;
            end if;
      when s4=>
            flag_df<='1';
            estado_sgte_df<=s5;
      when s5=>
            if tick_db='1' then
                  estado_sgte_df<=s0;
            end if;
      end case;
  end process;
--***************************Fin Bloque**********************************


  --************************Inicio Bloque**********************************
  --====Implementacion de un contador de flancos de subida cuya salida=========
  --====es actualizada en sincronia con tick_bt                      =========
  --====Las senales con terminacion _cf implican la pertenenencia    =========
--====a este bloque                                                 =========
  --====Senales de entrada: Bus2IP_Reset,Bus2IP_Clk,tick_bt1,tick_bt2 =========
  --====                    tick_df                                  =========
--====Senales internas  : cont_actual_cf,cont_sgte_cf              =========
  --====                  : pps_actual_cf,pps_sgte_cf                =========
  --====Senales de salida : pps_cf                                   =========
process(Bus2IP_Clk,Bus2IP_Reset,inicio)
  begin
    if inicio='1' or Bus2IP_Reset='1' then
      cont_actual_cf<=(others=>'0');
            pps_actual_cf<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
            cont_actual_cf<=cont_sgte_cf;
            pps_actual_cf<=pps_sgte_cf;
       end if;
  end process;

  cont_sgte_cf<=(others=>'0')    when tick_bt1='1' else
                cont_actual_cf+1 when tick_df='1'  else
                                  cont_actual_cf;

  pps_sgte_cf<=std_logic_vector(cont_actual_cf) when tick_bt2='1' else
                              pps_actual_cf;

  pps_cf<=pps_actual_cf;
```

```
--****************************Fin Bloque*************************************

  --************************Inicio Bloque*************************************
  --====Implementacion de una FSM que detecta flancos de subida       =========
  --====en ADQ(1)                                                    =========
  --====Las senales con terminacion _B implican la pertenenencia     =========
  --====a este bloque                                                =========
  --====Señales de entrada: Bus2IP_Reset,Bus2IP_Clk,ADQ(1)           =========
  --====Señales internas  : estado_actual_B,estado_sgte_B            =========
  --====Señales de salida : s_tick_B                                 =========
process(Bus2IP_Clk,Bus2IP_Reset)
  begin
       if Bus2IP_Reset='1' then
             estado_actual_B<=s0;
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
             estado_actual_B<=estado_sgte_B;
       end if;
  end process;

  process(estado_actual_B,ADQ(1))
begin
       estado_sgte_B<=estado_actual_B;
       s_tick_B<='0';
       case estado_actual_B is
       when s0=>
             if ADQ(1)='1' then
                   estado_sgte_B<=s1;
                   s_tick_B<='1';
             end if;
       when s1=>
             if ADQ(1)='0' then
                   estado_sgte_B<=s0;
             end if;
       end case;
  end process;
  --****************************Fin Bloque*************************************

--************************Inicio Bloque*************************************
  --====Implementacion de un circuito digital que obtiene el sentido  =========
  --====de giro a partir de dos señales en cuadratura de un encoder   =========
  --====Señales de entrada: ADQ(0),ADQ(1),tick_df,s_tickB,pps_cf      =========
  --====Señales internas  : try1,try2,try3,giro_sgte,giro_actual,     =========
  --====                  : giro                                      =========
  --====Señales de salida : tx                                       =========
process(Bus2IP_Clk,Bus2IP_Reset)
  begin
       if Bus2IP_Reset='1' then
             giro_actual<=(others=>'0');
       elsif Bus2IP_Clk'event and Bus2IP_Clk='1' then
             giro_actual<=giro_sgte;
       end if;
  end process;

  try1<=ADQ(0) and ADQ(1);
  try2<=try1 and tick_df;
  try3<=try1 and s_tick_B;

  giro_sgte<="00000010" when try2='1' else
```

```
                            "00000001" when try3='1' else
                            giro_actual;

giro<=giro_actual;
  tx<=pps_cf&giro;
  --****************************Fin Bloque***********************************

  --************************Inicio Bloque***********************************
  --===========Bloque de comunicacion de IP con el Procesador ================
  -- Obtencion de comando de procesador
  -- se recibe lo que el procesador envie, esto se alamcena en un  registro de
  -- 32 bits sincrono y puede ser leido desde rx, notese que este valor cambiara
  -- solo cada vez que BUS2IP_WrCE(0) sea 1 y luego de 1 ciclo de reloj.
lectura: process (Bus2IP_Clk,Bus2IP_Reset) is
  begin
    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
          if Bus2IP_Reset = '1' then
                rx_actual <= (others => '0');
              else
                rx_actual <= rx_sgte;
              end if;
        end if;
  end process lectura;

  rx_sgte<=Bus2IP_Data(0 to 31) when Bus2IP_WrCE(0) = '1' else
rx_actual;
  rx<=rx_actual;
  inicio<=rx(31);
  -- Envio de la lectura
  -- se envia lo que haya en tx de manera que es recomendable que tx sea
  -- la salida de algun registro sincrono.
IP2Bus_Data(0 to 31) <= tx when slv_read_ack = '1' else
                          (others => '0');

  slv_write_ack     <= Bus2IP_WrCE(0);
  slv_read_ack      <= Bus2IP_RdCE(0);
  IP2Bus_WrAck <= slv_write_ack;
  IP2Bus_RdAck <= slv_read_ack;
IP2Bus_Error <= '0';
  --====================Fin de Bloque de comunicacion ===================
--****************************Fin Bloque***********************************

end IMP;
```

Resumen del informe de la síntesis de la IP core ADQ

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

    Found finite state machine <FSM_0> for signal <estado_actual_dbA>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 5                                              |
    | Inputs             | 2                                              |
    | Outputs            | 2                                              |
    | Clock              | Bus2IP_Clk (rising_edge)                       |
    | Reset              | Bus2IP_Reset (positive)                        |
    | Reset type         | asynchronous                                   |
    | Reset State        | s0                                             |
    | Power Up State     | s0                                             |
    | Encoding           | automatic                                      |
    | Implementation     | automatic                                      |
    -----------------------------------------------------------------------
    Found finite state machine <FSM_1> for signal <estado_actual_dbB>.
    -----------------------------------------------------------------------
    | States             | 3                                              |
    | Transitions        | 5                                              |
    | Inputs             | 2                                              |
    | Outputs            | 2                                              |
    | Clock              | Bus2IP_Clk (rising_edge)                       |
    | Reset              | Bus2IP_Reset (positive)                        |
    | Reset type         | asynchronous                                   |
    | Reset State        | s0                                             |
    | Power Up State     | s0                                             |
    | Encoding           | automatic                                      |
    | Implementation     | automatic                                      |
    -----------------------------------------------------------------------
    Found finite state machine <FSM_2> for signal <estado_actual_dfA>.
    -----------------------------------------------------------------------
    | States             | 6                                              |
    | Transitions        | 10                                             |
    | Inputs             | 2                                              |
    | Outputs            | 2                                              |
    | Clock              | Bus2IP_Clk (rising_edge)                       |
    | Reset              | Bus2IP_Reset (positive)                        |
    | Reset type         | asynchronous                                   |
    | Reset State        | s0                                             |
    | Power Up State     | s0                                             |
    | Encoding           | automatic                                      |
    | Implementation     | automatic                                      |
    -----------------------------------------------------------------------
    Found finite state machine <FSM_3> for signal <estado_actual_dfB>.
    -----------------------------------------------------------------------
    | States             | 6                                              |
    | Transitions        | 10                                             |
    | Inputs             | 2                                              |
    | Outputs            | 2                                              |
    | Clock              | Bus2IP_Clk (rising_edge)                       |
    | Reset              | Bus2IP_Reset (positive)                        |
    | Reset type         | asynchronous                                   |
    | Reset State        | s0                                             |
```

```
    | Power Up State     | s0                                            |
    | Encoding           | automatic                                     |
    | Implementation     | automatic                                     |
    ----------------------------------------------------------------------
    Found finite state machine <FSM_4> for signal <estado_actual_bt>.
    ----------------------------------------------------------------------
    | States             | 4                                             |
    | Transitions        | 6                                             |
    | Inputs             | 2                                             |
    | Outputs            | 3                                             |
    | Clock              | Bus2IP_Clk (rising_edge)                      |
    | Reset              | Bus2IP_Reset (positive)                       |
    | Reset type         | asynchronous                                  |
    | Reset State        | s0                                            |
    | Power Up State     | s0                                            |
    | Encoding           | automatic                                     |
    | Implementation     | automatic                                     |
    ----------------------------------------------------------------------
    Found 26-bit up counter for signal <cont_actual_bt>.
    Found 12-bit up counter for signal <cont_actual_cf>.
    Found 10-bit up counter for signal <cont_actual_dbA>.
    Found 10-bit up counter for signal <cont_actual_dbB>.
    Found 8-bit register for signal <giro_actual>.
    Found 12-bit register for signal <pps_actual_cf>.
    Summary:
       inferred   5 Finite State Machine(s).
       inferred   4 Counter(s).
       inferred  20 D-type flip-flop(s).
Unit <user_logic> synthesized.
=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 9-bit subtractor                                      : 1
# Counters                                             : 4
 10-bit up counter                                     : 2
 12-bit up counter                                     : 1
 26-bit up counter                                     : 1
# Registers                                            : 36
 1-bit register                                        : 25
 12-bit register                                       : 1
 3-bit register                                        : 1
 32-bit register                                       : 4
 4-bit register                                        : 3
 8-bit register                                        : 1
 9-bit register                                        : 1
=========================================================================

=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================

Analyzing FSM <FSM_4> for best encoding.
Optimizing FSM <adq_felix_0/USER_LOGIC_I/estado_actual_bt/FSM> on signal
<estado_actual_bt[1:2]> with gray encoding.
-------------------
 State | Encoding
```

```
-------------------
 s0    | 00
 s1    | 01
 s2    | 11
 s3    | 10
-------------------
Analyzing FSM <FSM_3> for best encoding.
Optimizing FSM <adq_felix_0/USER_LOGIC_I/estado_actual_dfB/FSM> on signal
<estado_actual_dfB[1:3]> with gray encoding.
-------------------
 State | Encoding
-------------------
 s0    | 000
 s1    | 001
 s2    | 011
 s3    | 010
 s4    | 110
 s5    | 111
-------------------
Analyzing FSM <FSM_2> for best encoding.
Optimizing FSM <adq_felix_0/USER_LOGIC_I/estado_actual_dfA/FSM> on signal
<estado_actual_dfA[1:3]> with gray encoding.
-------------------
 State | Encoding
-------------------
 s0    | 000
 s1    | 001
 s2    | 011
 s3    | 010
 s4    | 110
 s5    | 111
-------------------
Analyzing FSM <FSM_1> for best encoding.
Optimizing FSM <adq_felix_0/USER_LOGIC_I/estado_actual_dbB/FSM> on signal
<estado_actual_dbB[1:2]> with gray encoding.
-------------------
 State | Encoding
-------------------
 s0    | 00
 s1    | 01
 s2    | 11
-------------------
Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <adq_felix_0/USER_LOGIC_I/estado_actual_dbA/FSM> on signal
<estado_actual_dbA[1:2]> with gray encoding.
-------------------
 State | Encoding
-------------------
 s0    | 00
 s1    | 01
 s2    | 11
-------------------
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# FSMs                                              : 5
# Adders/Subtractors                                : 1
```

```
 9-bit subtractor                                      : 1
# Counters                                             : 4
 10-bit up counter                                     : 2
 12-bit up counter                                     : 1
 26-bit up counter                                     : 1
# Registers                                            : 191
 Flip-Flops                                            : 191
=========================================================================

Device utilization summary:
---------------------------
Selected Device : 3s500efg320-4

 Number of Slices:                      124  out of   4656     2%
 Number of Slice Flip Flops:            149  out of   9312     1%
 Number of 4 input LUTs:                196  out of   9312     2%
 Number of IOs:                         203
 Number of bonded IOBs:                   0  out of    232     0%
---------------------------
Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 6.053ns (Maximum Frequency: 165.207MHz)
   Minimum input arrival time before clock: 3.040ns
   Maximum output required time after clock: 0.591ns
   Maximum combinational path delay: No path found
```

ANEXO G

Código VHDL para la obtención del sentido de giro a partir de dos señales en cuadratura de un encoder incremental

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity encoder is
    Port ( clk : in  STD_LOGIC;
           canal_A : in STD_LOGIC;
           canal_B : in STD_LOGIC;
           medicionA : out STD_LOGIC_VECTOR(15 DOWNTO 0);
           medicionB : out STD_LOGIC_VECTOR(15 DOWNTO 0);
           tickA : out STD_LOGIC;
           tickB : out STD_LOGIC;
           giro : out STD_LOGIC_VECTOR(1 DOWNTO 0);
           reset : in  STD_LOGIC);
end encoder;

architecture Behavioral of encoder is
type estado is(s0,s1);
--senales para procesamiento de canal A
signal estado_actual_A1,estado_sgte_A1 : estado;
signal estado_actual_A2,estado_sgte_A2 : estado;
signal cont_actual_A,cont_sgte_A : unsigned (15 DOWNTO 0);
signal s_tickA:std_logic;
--senales para procesamiento de canal B
signal estado_actual_B1,estado_sgte_B1 : estado;
signal estado_actual_B2,estado_sgte_B2 : estado;
signal cont_actual_B,cont_sgte_B : unsigned (15 DOWNTO 0);
signal s_tickB:std_logic;
signal try1,try2,try3:std_logic;
signal giro_actual,giro_sgte:STD_LOGIC_VECTOR(1 DOWNTO 0);
begin
--====================registro de estados====================
process(clk,reset)
begin
       if reset='1' then
              estado_actual_A1<=s0;
              estado_actual_A2<=s0;
              cont_actual_A<=(others=>'0');
              estado_actual_B1<=s0;
              estado_actual_B2<=s0;
              cont_actual_B<=(others=>'0');
              giro_actual<=(others=>'0');
```

```
	elsif clk'event and clk='1' then
		estado_actual_A1<=estado_sgte_A1;
		estado_actual_A2<=estado_sgte_A2;
		cont_actual_A<=cont_sgte_A;
		estado_actual_B1<=estado_sgte_B1;
		estado_actual_B2<=estado_sgte_B2;
		cont_actual_B<=cont_sgte_B;
		giro_actual<=giro_sgte;
	end if;
end process;
--============logica del estado siguiente para canal A===========
process(estado_actual_A1,canal_A)
begin
	estado_sgte_A1<=estado_actual_A1;
	s_tickA<='0';
	case estado_actual_A1 is
	when s0=>
		if canal_A='1' then
			estado_sgte_A1<=s1;
			s_tickA<='1';
		end if;
	when s1=>
		if canal_A='0' then
			estado_sgte_A1<=s0;
		end if;
	end case;
end process;

process(s_tickA,estado_actual_A2)
begin
	estado_sgte_A2<=estado_actual_A2;
	case estado_actual_A2 is
	when s0=>
		if s_tickA='1' then
			estado_sgte_A2<=s1;
		end if;
	when s1=>
		estado_sgte_A2<=estado_actual_A2;
	end case;
end process;

process(s_tickA,cont_actual_A,estado_actual_A2)
begin
	case estado_actual_A2 is
	when s0=>
		cont_sgte_A<=(others=>'0');
	when s1=>
		if s_tickA='1' then
			cont_sgte_A<=(others=>'0');
		else
			cont_sgte_A<=cont_actual_A+1;
		end if;
	end case;
end process;

tickA<=s_tickA;

medicionA<=std_logic_vector(cont_actual_A);
```

```
--============logica del estado siguiente para canal B===========
process(estado_actual_B1,canal_B)
begin
	estado_sgte_B1<=estado_actual_B1;
	s_tickB<='0';
	case estado_actual_B1 is
	when s0=>
		if canal_B='1' then
			estado_sgte_B1<=s1;
			s_tickB<='1';
		end if;
	when s1=>
		if canal_B='0' then
			estado_sgte_B1<=s0;
		end if;
	end case;
end process;

process(s_tickB,estado_actual_B2)
begin
	estado_sgte_B2<=estado_actual_B2;
	case estado_actual_B2 is
	when s0=>
		if s_tickB='1' then
			estado_sgte_B2<=s1;
		end if;
	when s1=>
		estado_sgte_B2<=estado_actual_B2;
	end case;
end process;

process(s_tickB,cont_actual_B,estado_actual_B2)
begin
	case estado_actual_B2 is
	when s0=>
		cont_sgte_B<=(others=>'0');
	when s1=>
		if s_tickB='1' then
			cont_sgte_B<=(others=>'0');
		else
			cont_sgte_B<=cont_actual_B+1;
		end if;
	end case;
end process;

tickB<=s_tickB;
medicionB<=std_logic_vector(cont_actual_B);

try1<=canal_A and canal_B;
try2<=try1 and s_tickA;
try3<=try1 and s_tickB;

giro_sgte<=  "01" when try2='1' else
             "10" when try3='1' else
             giro_actual;
giro<=giro_actual;
end Behavioral;
```

BIBLIOGRAFÍA

[1] Fei-Yue Wang, Derong Liu, "Networked Control System, Theory and Application", Springer – Estados Unidos de América, 2008.

[2] Página web de la empresa aJile System: www.ajile.com

[3] Página web de la empresa Comedi: www.comedi.com

[4] Eduardo Boemo Scalvinoni, "Cuaderno Tecnológico Nº 1: Estado del arte en la tecnología FPGA", Instituto Nacional de Tecnología Industrial – Argentina, 2005.

[5] Imran Ahmed, Hong Wong y Vikram Kapila, "Internet-Based Remote Control using a Microcontroller and an Embedded Ethernet", Department of Mechanical, Aerospace, and Manufacturing Engineering Polytechnic University – Estados Unidos de América.

[6] Andrés Felipe Ruiz Olaya, "Implementación de una red MODBUS/TCP", Universidad del Valle – Colombia, 2002.

[7] Luis Acosta, José Granados, Jorge Duque y Sergio Fiallo, "Diseño e Implementación de una RTU en FPGA" Universidad Tecnológica de Bolivar – Venezuela

[8] Spartan-3 Generation FPGA User Guide UG331 (v1.5).Xilinx, 2009.

[9] Information Sciences Institute, University of Southern California, "Internet Protocol RFC791", Defense Advanced Research Projects Agency Information Processing Techniques Office – Estados Unidos de América, 1981.

[10] Documentación XPS Ethernet Lite Media Access Controller (v2.00a)

[11] M. Simmons. "Ethernet Theory of Operation AN1120". Microchip Technology Inc, 2008.

[12] J. Postel, "User Datagram Protocol RFC768", Information Sciences Institute, University of Southern California – Estados Unidos de América, 1980.

[13] Documentación XPS Interrupt Controller v1.00a

[14] Documentación XPS Timer/Counter v1.00a

[15] Documentación API, Driver XEmacLite

[16] Documentación API, Driver XPS Interrupt Controller

[17] Documentación API, Driver XPS Timer/Counter

[18] Peter C. Dibble. “Real Time Java Programming”. Prentice Hall PTR – Estados Unidos de América, 2002.

[19] Ramu Krishnan. “Electric Motor Drive”. Prentice Hall PTR – Estados Unidos de América, 2002.

[20] Hoja de datos LAN83C185. SMSC, 2004

[21] Spartan-3E Starter Kit Board User Guide UG230, Xilinx, 2009.

[22] Spartan-3 Generation Configuration User Guide UG332 (v1.5).Xilinx, 2009.

[23] Página Web del programa Xilinx University, http://www.xilinx.com/university/workshops/embedded-system-design-flow/index.htm

[24] Hoja de datos L298. ST Microelectronics. 2000.

[25] Hoja de datos H11L1. Fairchild Semiconductor. 2005

[26] Msc Ing. Eleazar Sal y Rosas. “Curso de Telecontrol”. Universidad Nacional de Ingeniería, 2010.

[27] Greg Bollela. “The Real-Time Specification for Java”. Addison-Wesley – Estados Unidos de América, 2000.

[28] Umberto Salsi, www.icosaedro.it.

[29] FPGA Skills Acquisition, Part II - “The Embedded Internet”, Laboratory - Session I. (Xilinx EDK 8.1).

[30] FPGA Skills Acquisition, Part II - “The Embedded Internet”, Laboratory - Session II. (Xilinx EDK 8.1).

[31] Antonio Díaz Estrella. “Teoria y diseño con microcontroladores de Freescale”. Universidad de Málaga – España, 2008.

[32] Hoja de datos 40106. Fairchild Semicoductor. 2003.

[33] Javier García de Jalón. “Aprenda Java como si estuviera en primero”. Universidad de Navarra, Marzo 1999.

[34] Bruce Eckel. “Thinking in Java”. Prentice Hall PTR, 1998.

[35] Katsuhiko Ogatha, “Sistemas de control en tiempo discreto”, University of Minnesota – Estados Unidos de América, 1996.

[36] Katsuhiko Ogatha, “Sistemas de control en tiempo continuo”, University of Minnesota – Estados Unidos de América, 1998.