ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA.

CONTROL DEL MÓDULO PENDUBOT UTILIZANDO UNA FPGA

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL.

RODRIGO ALEXANDER LÓPEZ ENCALADA
Rodrigo_alexanderlopez@hotmail.com

DIRECTOR: PATRICIO CHICO, MSc
Patricio.chico@epn.edu.ec

Quito, AGOSTO 2011

DECLARACIÓN

Yo Rodrigo Alexander López Encalada, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Rodrigo Alexander López Encalada

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por el señor Rodrigo Alexander López Encalada, bajo mi supervisión.

_____

MSc. Patricio Chico.

DIRECTOR DE PROYECTO

AGRADECIMIENTO

Nunca un año se presentó con tantas pruebas y obstáculos, con seguridad puedo decir que los aprendizajes obtenidos en este proceso marcarán mi camino de hoy en adelante. Quiero Agradecer a Dios, por darme fortaleza para vencer los retos que se me han presentado y permitirme culminar una etapa más en mi vida.

A mis padres Rodrigo y Alexandra, por la confianza y apoyo recibido hasta el final de mi carrera universitaria. A mis hermanos Viviana, Guido, Belén, Paula y Nicole, por su constante apoyo, entusiasmo y comprensión en cada semestre de estudio.

Al MSc. Patricio Chico por su respaldo, colaboración y apoyo constante para la elaboración del presente proyecto.

A mis amigos y compañeros de mil batallas que siempre han estado apoyándome en altos y bajos.

Finalmente un agradecimiento muy sincero a la Escuela Politécnica Nacional especialmente todos los profesores que cuyos conocimientos y experiencias fueron de vital importancia para mi formación universitaria.

Rodrigo

DEDICATORIA

Definitivamente debo dedicar este proyecto a todos aquellos que estuvieron, los que están y continúan a mi lado, a mis amigos y especialmente a mi familia.

Rodrigo

CONTENIDO

RESUMEN

Hoy en día, las FPGA (*Field Programmable Gate Array*) aparecen en dispositivos como instrumentos electrónicos, automóviles, artefactos aéreos, máquinas copiadoras, hardware computacional de aplicaciones específicas, en productos de control industrial, etc. La funcionalidad FPGA no ha sido previamente accesible para ingenieros de control industrial. Programar las FPGA ha sido limitado a personas con experiencia y conocimiento profundo en los HDL u otras herramientas de diseño de bajo nivel las cuales requieren de un alto grado de profundidad en el aprendizaje.

En este trabajo se propone desarrollar un algoritmo de control para el Pendubot (sistema no lineal subactuado) utilizando la capacidad de procesamiento de un FPGA (*Field Programmable Gate Array*), en la cual se implementan los algoritmos de adquisición de datos, pre-procesado, algoritmo de control, regulación del punto de operación y determinación de la salida del sistema, así como también el módulo de comunicación con un computador para registrar los valores de las variables a controlar.

El algoritmo de adquisición y control se desarrollará en un lenguaje de alto nivel, utilizando el Módulo FPGA de LabVIEW y se descarga en la tarjeta Spartan-3E Starter Kit. Con el Módulo FPGA de LabVIEW, se ha abierto la tecnología FPGA a un mayor rango de estudiantes que ahora pueden definir la lógica FPGA usando el desarrollo gráfico de LabVIEW. Los estudiantes de control y medición ahora pueden enfocarse primordialmente en las aplicaciones de prueba y control, donde se debe insistir en la práctica.

Es importante señalar que el Pendubot mencionado se encuentra previamente construido, y se adaptará el hardware existente para ser controlado por la FPGA.

PRESENTACIÓN

En el presente proyecto se pretende abordar una nueva tecnología, FPGA, para ser utilizado en el control de sistemas, programada a través del software LABVIEW FPGA. Para su presentación se resuelve el problema de regulación del Pendubot.

La Spartan-3E es una FPGA (Field Programmable Gate Array, Arreglo de Compuertas Programables en Campo) que por su bajo coste y facilidad de programación permite que sea asequible para proyectos que vayan orientados a la ingeniería robótica, informática e industria. Consta principalmente de periféricos ya conocidos, como: puertos VGA, PS/2, USB, LCD, Switch, Pulsadores, Ethernet, etc

Las FPGA, fueron inventadas en 1984 por Ross Freeman y Bernard Vonderschmitt co-fundadores de Xilinx, y surgen como evolución de los CPLDs [1]

Una FPGA es un dispositivo que está constituido por bloques de lógica interconectados cuya funcionalidad se puede programar. Los circuitos son plenamente reprogramables y, por lo tanto, son muy flexibles (en términos del rango de diseños prácticos para los cuales pueden ser usadas), son de fácil uso y programación y ofrecen un gran rendimiento.

Las FPGA ejecutan los bloques internos de forma concurrente, es decir completamente en paralelo, se puede crear un gran número de tareas específicas que se ejecutan como circuitos en paralelos dentro del chip.

Los microprocesadores, microcontroladores y DSP’s carecen de esta funcionalidad ya que están forzados a procesar una instrucción a la vez utilizando esquemas complejos (manejo de hilos u otros) para lograr un pseudoparalelismo,

e inclusive con nuevas tecnologías como los procesadores multi-nucleo no pueden lograr un procesamiento en paralelo como las FPGA.

Finalmente a diferencia de los ASIC (Circuito Integrado para Aplicaciones Específicas), los FPGA son reprogramables, actualizar un sistema con alguna corrección en el diseño es posible mientras que con los ASIC se tiene un diseño fijo y hacer un cambio es demasiado complejo.

El Pendubot fue desarrollado originalmente en “Coordinated Science Laboratory, University of Illinois at Urbana Champaign” bajo la dirección del Profesor Mark W. Spong. El Pendubot, nombre formado por la contracción de las palabras PENDULUM y ROBOT, es un sistema electromecánico subactuado diseñado para educación, investigación e implementación de algoritmos de control.

El Pendubot tiene la estructura de un robot manipulador de dos grados de libertad de enlaces rígidos, pero no tiene actuador en la segunda articulación, y sólo está restringido por el movimiento de la primera articulación debido al acoplamiento que existe entre ellas.

Algunos trabajos relacionados al respecto son Block y Spong [2] usando la técnica de Swing Up y estabilizando el Pendubot por medio de un controlador lineal LQR (Regulador Cuadrático Lineal). El reto de este mecanismo es que, en ausencia de control, las posiciones deseadas son puntos inestables del sistema.

Los capítulos se encuentran distribuidos de la siguiente manera.

El Capítulo Uno “Fundamentos Básicos”, se hace una breve descripción del sistema, objetivos de control y el sistema de control.

El Capítulo Dos “Modelado Matemático del Pendubot”, se presenta todo el modelo matemático superficialmente, debido a que su análisis a profundidad se encuentra en *Andrade S. Renato German,* “*Análisis, Diseño y Construcción del*

PENDUBOT" [3], contiene el modelado dinámico, los puntos de equilibrio y su aproximación lineal.

El Capítulo Tres "Diseño de los Controladores del Sistema Pendubot" se obtienen los controladores necesarios para el funcionamiento del sistema Pendubot; se tiene un controlador diferente para cada posición del Pendubot, esto es el LQR para el equilibrio mientras que para el balanceo, que es llevar el sistema al punto cercano de funcionamiento se realiza una linealización parcial por retroalimentación de estados y se construye un controlador PD para trazar la trayectoria que debe seguir el eslabón uno y llevarlo a la posición deseada y con ello también llevar al eslabón dos.

El Capítulo Cuatro "Descripción del FPGA", se describe la FPGA, su tecnología, arquitectura, evolución y aplicaciones, luego se detalla el módulo Spartan-3E Starter Kit, el cual es el que se utiliza para este proyecto, se describen los componentes que se utilizan y en el Anexo C se detallan todos los componentes del módulo. También se describe su programación en LABVIEW.

El Capítulo Cinco "Implementación del Algoritmo" se detalla la programación del control, así como también la lógica de conmutación de cada controlador.

El Capítulo Seis "Pruebas y Resultados" muestra la ejecución del Algoritmo desde la interfaz gráfica y también su ejecución sin interfaz gráfica.

Finalmente, el Capítulo Siete "Conclusiones y Recomendaciones", demuestra la utilización de nueva tecnología en sistemas de control, así como también la utilización de software adicional para una mejor y fácil abstracción del algoritmo.

CAPÍTULO 1

FUNDAMENTOS BÁSICOS

Los beneficios que la ingeniería de control aporta a la sociedad actual son indiscutibles, ayudando significativamente a la mejora de la calidad de los productos fabricados, al aumento de la eficiencia de los procesos, a la minimización del consumo de energía, al aumento de la seguridad de las plantas industriales entre otros [4].

En la mayoría de los sistemas mecánicos, como robots articulados de enlaces rígidos, existe una fuerza independiente actuando sobre cada articulación. Existen una clase de sistemas, donde una fuerza independiente (actuador) está ausente de por lo menos una de las articulaciones, a este tipo de sistemas se le conoce como sistemas mecánicos subactuados [4].

Los sistemas mecánicos subactuados son sistemas físicos no lineales de gran complejidad y se caracterizan porque las técnicas de control convencionales son ineficaces cuando se requiere que ellos estén en un punto de operación o posición de equilibrio [5]. Ante esta condición se requiere la implementación de técnicas de control para sistemas no lineales, las cuales permiten manejar el sistema y adaptarse a los cambios en los puntos de equilibrio de la planta.

Un sistema ampliamente usado es el Pendubot, debido a que permite ilustrar la aplicación, validez y desempeño de diferentes técnicas de control y su implementación en diferentes dispositivos, además que representa un banco de pruebas adecuado.

Diseño de controladores basados en plataformas reconfigurables que permitan una mayor flexibilidad en la arquitectura y que operen de manera adecuada para sistemas subactuados han sido objeto de estudio desde hace varios años, las características de flexibilidad y reconfigurabilidad de los dispositivos de lógica programable tipo FPGA, resultan atractivos como plataforma de diseño de hardware digital de bajo costo.

El acelerado desarrollo de nuevas tecnologías y la continua demanda de sistemas de control que exigen mayor precisión, rapidez, escalabilidad y fácil programación, hacen necesarios manipular tecnología avanzada como son las FPGA que permiten un mejor control para tareas dedicadas.

En la actualidad las FPGA están siendo utilizadas ampliamente para tareas de control, donde se las ha aprovechado al máximo debido a su naturaleza reprogramable.

1.1 SISTEMAS SUBACTUADOS.

Son aquellos que poseen un número menor de actuadores que grados de libertad. La ausencia de un actuador representa un reto en el diseño de estrategias de control.

Los sistemas subactuados surgen de varias formas [6]:

- Por diseño.- Los sistemas subactuados por diseño más comunes son prototipos de laboratorio, como por ejemplo el robot “brachiation” de Fukuda, el Acrobot, el péndulo invertido sobre un carro y el péndulo invertido sobre un brazo rotatorio, el Pendubot y más recientemente el péndulo con volante de inercia. Estos sistemas son robots multieslabones que tienen motores (actuadores) sólo en algunas de las uniones.
- Sistemas móviles.- Existen una gran variedad de sistemas subactuados móviles, como por ejemplo: carros, satélites, submarinos, barcos. Sin embargo, se tienen sistemas subactuados móviles más complejos al colocar un brazo manipulador montado sobre una plataforma móvil, por ejemplo una plataforma espacial o un vehículo submarino. Aquí las fuerzas de reacción debido al movimiento del brazo están acopladas a la plataforma que no puede ser considerada como un punto de referencia inercial. Para este tipo de sistemas se usan controles adicionales para mantener el control de posición de la base. Sin embargo, si estos

actuadores fallan o son apagados para ahorrar energía, entonces el sistema completo es subactuado.

- Por modelo.- Una tercera forma en que surgen los sistemas subactuados es debida al modelo matemático que se esté usando para el diseño del controlador; por ejemplo, cuando la flexibilidad de la unión se incluye en el modelo.

Los sistemas subactuados son sistemas en los que no es posible realizar un control directo sobre el grado de libertad no actuado lo que hace necesario implementar técnicas de control especiales para este tipo de sistemas. Este es un campo de gran interés para la teoría de control moderna.

1.2 PENDUBOT.

Un sistema subactuado ampliamente usado es el Pendubot, debido a que permite ilustrar la aplicación, validez y desempeño de diferentes técnicas de control y su implementación en diferentes dispositivos electrónicos, además que representa un banco de pruebas adecuado.

1.2.1 DESCRIPCIÓN DEL SISTEMA

En esta sección se describirá al sistema PENDUBOT de una manera somera, particularizando las posiciones de equilibrio y los elementos de censado. La información de este sistema se basa en gran medida en la siguiente fuente, *Andrade S. Renato German,* "*Análisis, Diseño y Construcción del PENDUBOT"* [3].

El Pendubot, nombre formado por la contracción de las palabras PENDULUM y ROBOT, es un sistema electromecánico que consiste de dos eslabones rígidos interconectados por uniones o junturas rotacionales, el cual tiene la estructura de un robot manipulador de dos grados de libertad, tal como se muestra en la figura 1.1.

- El primer eslabón está actuada por un motor de DC y el segundo eslabón no está actuada, es decir que se puede mover libremente.

- Tiene una gran variedad de modos de comportamiento.

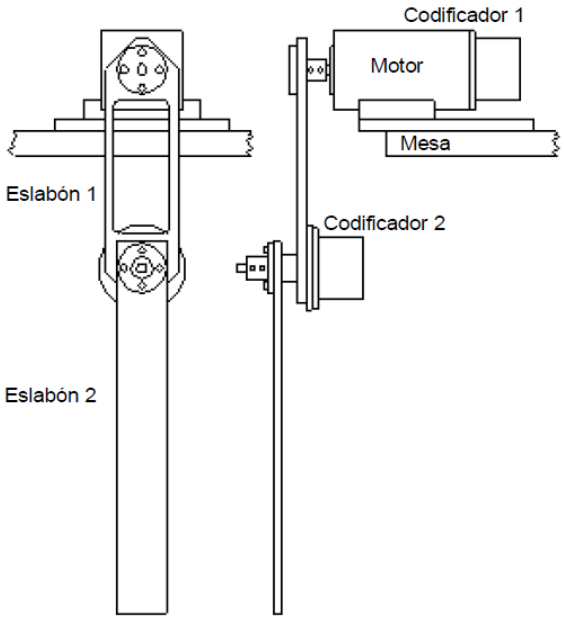
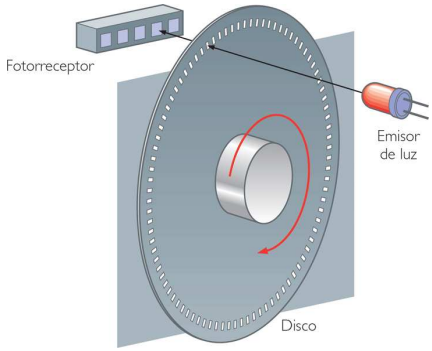


Figura 1.1 Pendubot [6]

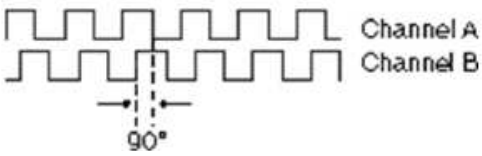
El eslabón uno está acoplado directamente al eje del motor DC de imán permanente de 72 V y posee además el alojamiento para el eje del segundo eslabón, el cual se mueve libremente y es controlado por acción del primer eslabón.

Para la medición de la posición y velocidad angular de cada enlace se emplean codificadores (encoder) ópticos incrementales de 1024 [*pulsos/revolución*].

Los encoder óptico incrementales, figura 1.2 (a), proporcionan generalmente dos formas de ondas cuadradas y desfasadas entre si 90^{o} eléctricos (en cuadratura), los cuales son “canal A” y “canal B”, figura 1.2 (b), con la lectura de un solo canal, canal A, se dispone de la información correspondiente a la posición y velocidad de rotación, mientras que si se capta también la otra señal, canal B, se tiene el sentido de giro.



(a)



(b)

Figura 1.2 (a) Representación Encoder Óptico [7]

(b) Pulsos de salida de un encoder desfasados 90^{o} [8]

La posición angular del enlace uno se mide con respecto a un sistema fijo de coordenadas $0_{x_0 y_0 z_0}$ ubicado sobre el eje del enlace uno, y del enlace dos con respecto a un eje que constituye la prolongación del enlace uno, tal como se muestra en la figura 1.3

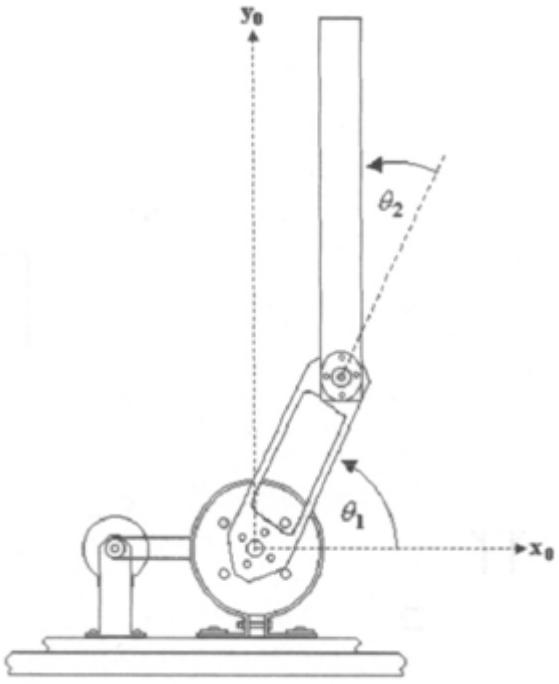


Figura 1.3. Posición angular [3]

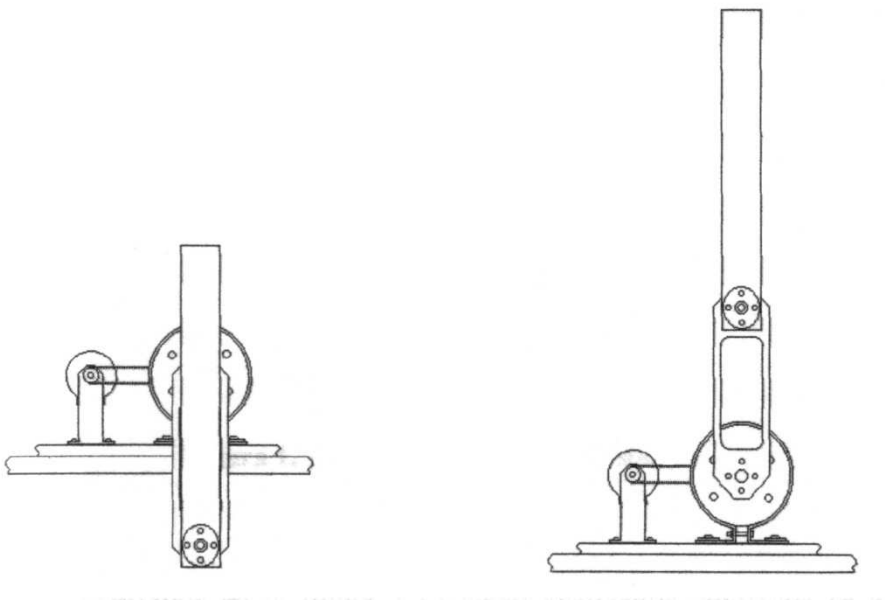
De acuerdo al modelo dinámico, el sistema cuenta con cuatro puntos de equilibrio principales con entrada cero $(\tau = 0)$, uno estable $\left(-\frac{\pi}{2}, 0\right)$ y tres inestables $\left(\frac{\pi}{2}, 0\right)$, $\left(-\frac{\pi}{2}, \pi\right)$ y $\left(\frac{\pi}{2}, \pi\right)$

1.2.2 OBJETIVO DE CONTROL.

En este sistema los objetivos básicos de control son:

- **Control de Balanceo**, el cual consiste en llevar los enlaces desde su punto de equilibrio estable $(-\frac{\pi}{2}, 0)$, a sus puntos de equilibrio inestables: posición tope $\left(\frac{\pi}{2}, 0\right)$ y su posición media $\left(-\frac{\pi}{2}, \pi\right)$, a través de los movimientos del primer eslabón.
- **Control de Equilibrio**, que consiste en hacer estable los puntos inestables del sistema.
- **Seguimiento**, en el cual se trata de que el sistema una vez estabilizado, siga una referencia de posición variante en el tiempo.

Las posiciones inestables de equilibrio se ilustran en la figura 1.4.



(a) **(b)**

Figura 1.4 (a) Posición media **(b)** Posición tope [3]

Uno de los problemas de control que presenta el sistema Pendubot, es que su modelo matemático es altamente no lineal y por lo tanto es difícil encontrar una ley de control única que resuelva los tres objetivos antes mencionados.

1.3 SISTEMA DE CONTROL

La tendencia actual de diseño de circuitos integrados es que estos realicen todas las funciones que se requieran en un solo chip, por tal motivo se implementan en la FPGA los algoritmos de adquisición de datos, pre-procesado, algoritmo de control, regulación del punto de operación y determinación de la salida del sistema, así como también el módulo de comunicación con un computador para registrar los valores de las variables a controlar.

Para esto se utiliza la tarjeta Spartan-3E Starter Kit para que realice todas las funciones que son necesarias para el correcto funcionamiento del sistema.

La figura 1.5 muestra la tarjeta Spartan-3E Starter Kit de Xilinx

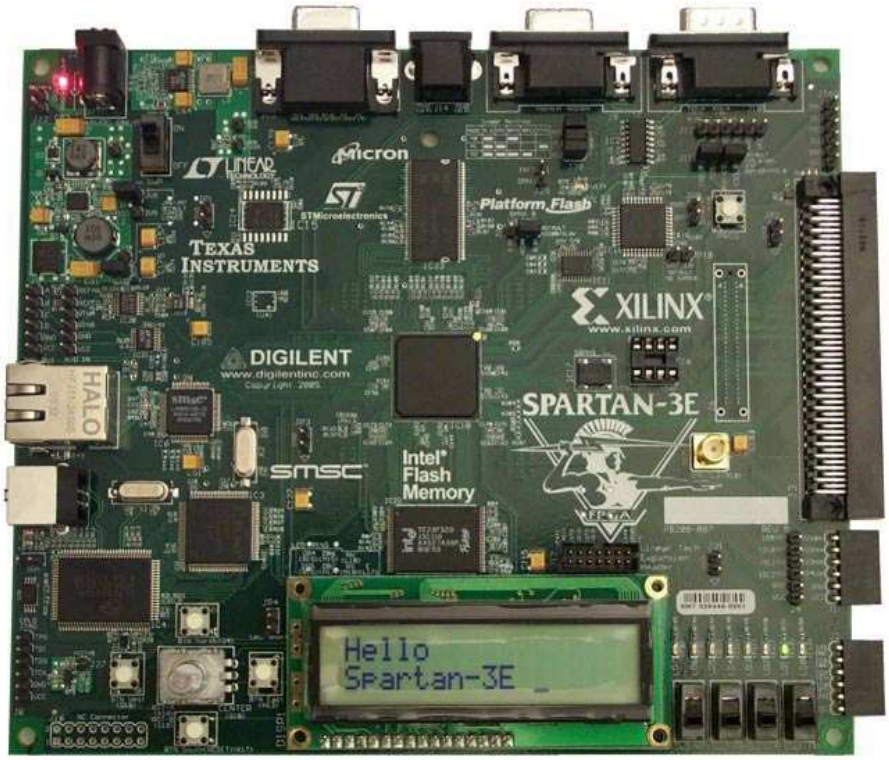


Figura 1.5 Spartan-3E de Xilinx [9]

En la figura 1.6 se muestra el diagrama de control que generará la salida de control a partir de las entradas generadas por los sensores en el sistema.
El sistema tiene cuatro entradas, dos entradas para cada encoder incremental que están en cada enlace, los leds indicadores del sistema, los pulsadores y switch de control, la comunicación con la PC y la salida del control.

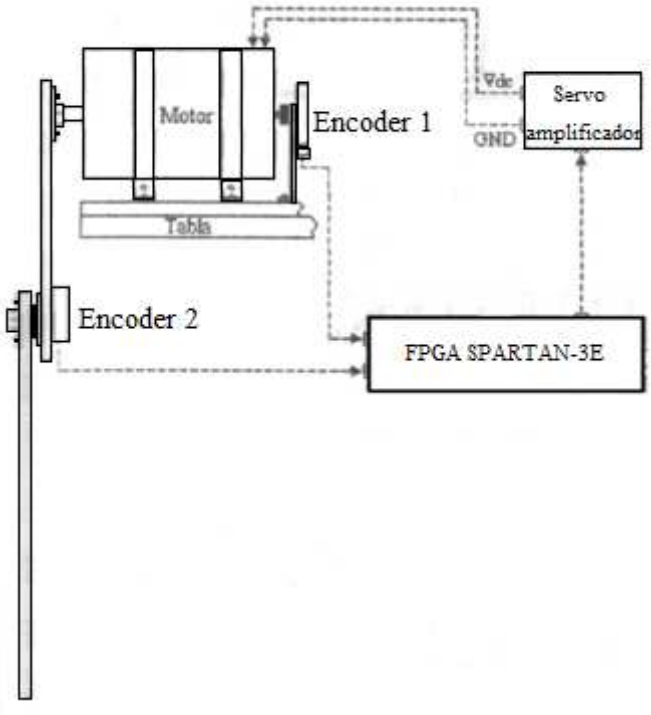


Figura 1.6 Diagrama de control

El sistema de control diseñado trabaja en base al modelo matemático. La obtención de un buen modelado matemático del sistema electromecánico subactuado Pendubot es importante para predecir su funcionamiento, ya que la validez de la predicción y el comportamiento de las estructuras de control diseñadas depende, en gran medida, del modelo dinámico utilizado [6].

CAPÍTULO 2

MODELADO MATEMÁTICO DEL PENDUBOT

El *Pendubot* es un sistema mecánico subactuado, que fue creado para fines didácticos con el fin de ilustrar los diferentes conceptos de sistemas no lineales, robótica y teoría de sistemas de control.

El sistema electromecánico subactuado puede verse como un péndulo invertido, en el que el motor hace las veces del carro. Este sistema es difícil de controlar debido a que el segundo eslabón está libre y que su modelo es de fase no mínima, es decir, que el sistema tiene polos y ceros en el semiplano positivo.

El modelo dinámico del sistema electromecánico subactuado, se hace sin considerar la fricción ya que este modelo se utiliza en todo el trabajo.

2.1 MODELO DINÁMICO DEL PENDUBOT

De acuerdo a la naturaleza del sistema, se puede obtener las ecuaciones diferenciales no lineales invariantes en el tiempo, las cuales determinarán el comportamiento dinámico.

Estas ecuaciones se las obtendrá por el método de Euler-Lagrange, que está básicamente enfocada en la descripción de la energía del sistema.

La determinación del modelo físico se la obtiene completando los siguientes pasos:

1. Determinación de todos los parámetros físicos del Pendubot
2. Determinación de las ecuaciones no-lineales del sistema, utilizando el método de Euler-Lagrange.
3. Linealización del sistema no-lineal.

2.1.1 DETERMINACIÓN DE PARÁMETROS

La figura 2.1 muestra un diagrama esquemático del Pendubot.

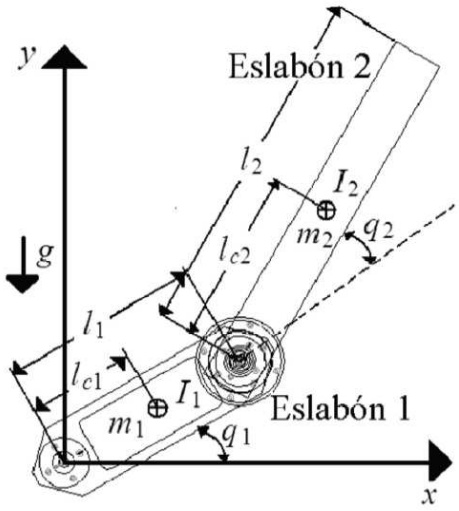


Figura 2.1 Esquema del Pendubot [4]

Donde:

- q_i, desplazamiento angular del eslabón i.
- m_i, masa en kg del enlace i.
- l_i, longitud del enlace i en metros.
- l_{ci}, distancia del centro de masa del enlace i.
- I_i, momento de inercia del enlace i.

Los parámetros han sido tomados de [3], y se muestran a continuación:

$$m_1 = 0{,}17974\ Kg$$
$$m_2 = 0{,}26269\ Kg$$
$$l_1 = 0{,}151\ m$$
$$l_{c1} = 0{,}09\ m$$
$$l_{c2} = 0{,}148\ m$$
$$I_{zz1} = 0{,}0010624\ Kg\ m^2$$
$$I_{zz2} = 0{,}002537\ Kg\ m^2$$

2.1.2 DETERMINACIÓN DEL MODELO NO LINEAL

De acuerdo a la ecuación de Euler-Lagrange, el sistema puede ser descrito de la siguiente manera:

$$Q_i = \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{q}_i}\right) - \frac{\delta L}{\delta q_i} \qquad \text{(i=1, 2… n)} \qquad (2.1)$$

Donde el Lagrangiano *L* está formado por la diferencia entre la energía cinética *K* y la energía potencial *V*, esto es:

$$L = K\text{-}V \qquad (2.2)$$

En el caso del Pendubot, las coordenadas q_i, representan las posiciones angulares de los enlaces, mientras que Q_i representa el vector de torques del sistema, con lo cual la expresión (2.1) queda:

$$\tau_i = \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\theta}_i}\right) - \frac{\delta L}{\delta \theta_i} \qquad (2.3)$$

Con esta expresión se obtiene la dinámica del sistema.

La energía cinética del sistema es la suma de las energías cinéticas de los eslabones uno y dos, esto es:

$$K = \sum_{i=1}^{2} k_i = k_1 + k_2$$

Donde:

k_i Es la energía cinética del eslabón i.

$$k_1 = \frac{1}{2} m_1 {l_{c1}}^2 {\dot{\theta}_1}^2 + \frac{1}{2} I_{zz1} {\dot{\theta}_1}^2 + \frac{1}{2} I_{zz2} (\dot{\theta}_1 + \dot{\theta}_2)^2$$

$$k_2 = \frac{1}{2} m_2 \left[({l_1}^2 + 2 l_1 l_{c2} cos\theta_2 + {l_{c2}}^2) {\dot{\theta}_1}^2 + (2 l_1 l_{c2} cos\theta_2 + 2 {l_{c2}}^2) \dot{\theta}_1 \dot{\theta}_2 + {l_{c2}}^2 {\dot{\theta}_2}^2 \right]$$

Por lo tanto:

$$K = \frac{1}{2} m_2 \left[({l_1}^2 + 2 l_1 l_{c2} cos\theta_2 + {l_{c2}}^2) {\dot{\theta}_1}^2 + (2 l_1 l_{c2} cos\theta_2 + 2 {l_{c2}}^2) \dot{\theta}_1 \dot{\theta}_2 + {l_{c2}}^2 {\dot{\theta}_2}^2 \right] + \frac{1}{2} m_1 {l_{c1}}^2 {\dot{\theta}_1}^2 + \frac{1}{2} I_{zz1} {\dot{\theta}_1}^2 + \frac{1}{2} I_{zz2} (\dot{\theta}_1 + \dot{\theta}_2)^2$$

La energía potencial del sistema es la suma de la energía potencial de cada enlace, es decir:

$$V = \sum_{i=1}^{2} v_i = v_1 + v_2$$

Donde:

v_i Es la energía potencial del eslabón i.

$v_1 = m_1 g l_{c1} sen\theta_1 + m_2$

$v_2 = m_2 g[l_1 sen\theta_1 + l_{c2} sen(\theta_1 + \theta_2)]$

Por lo tanto:

$$V = m_1 g l_{c1} sen\theta_1 + m_2 g[l_1 sen\theta_1 + l_{c2} sen(\theta_1 + \theta_2)]$$

Una vez determinadas las expresiones de energías, se reemplazará en (2.2) para obtener el Lagrangiano L.

$$\begin{aligned} L = & \left[\frac{1}{2} m_2 \left[(l_1^{\,2} + 2 l_1 l_{c2} cos\theta_2 + l_{c2}^{\,2})\dot{\theta}_1^{\,2} + (2 l_1 l_{c2} cos\theta_2 + 2 l_{c2}^{\,2})\dot{\theta}_1\dot{\theta}_2 + l_{c2}^{\,2}\dot{\theta}_2^{\,2}\right]\right. \\ & \left. + \frac{1}{2} m_1 l_{c1}^{\,2}\dot{\theta}_1^{\,2} + \frac{1}{2} I_{zz1}\dot{\theta}_1^{\,2} + \frac{1}{2} I_{zz2}(\dot{\theta}_1 + \dot{\theta}_2)^2\right] \\ & - [m_1 g l_{c1} sen\theta_1 + m_2 g[l_1 sen\theta_1 + l_{c2} sen(\theta_1 + \theta_2)]] \end{aligned} \tag{2.4}$$

Reemplazando los valores de (2.4) dentro de la ecuación de Euler-Lagrange (2.3) y haciendo los cálculos necesarios en varios pasos se obtienen las ecuaciones dinámicas del sistema τ_1 y τ_2, donde $\tau_2 = 0$, porque el enlace dos no tiene actuador. (Para mayor detalle en la obtención de las ecuaciones dinámicas referirse a [3]).

Las ecuaciones dinámicas del sistema son:

$$\tau_1 = \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\theta}_1}\right) - \frac{\delta L}{\delta \theta_1}$$

$$\begin{aligned} \tau_1 = & (m_1 l_{c1}^{\,2} + m_2 l_1^{\,2} + 2 m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}^{\,2} + I_{zz1} + I_{zz2})\ddot{\theta}_1 \\ & - 2 m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_1 \dot{\theta}_2 - m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_2^{\,2} \\ & + (m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}^{\,2} + I_{zz2})\ddot{\theta}_2 + m_1 g l_{c1} cos\theta_1 \\ & + m_2 g l_{c2} cos(\theta_1 + \theta_2) + m_2 g l_1 cos\theta_1 \end{aligned} \tag{2.5}$$

$$\tau_2 = 0 = \frac{d}{dt}\left(\frac{\delta L}{\delta \dot{\theta}_2}\right) - \frac{\delta L}{\delta \theta_2}$$

$$0 = \left(m_2 l_1 l_{c2} cos\theta_2 + m_2 {l_{c2}}^2 + I_{zz2}\right)\ddot{\theta}_1 + \left(m_2 {l_{c2}}^2 + I_{zz2}\right)\ddot{\theta}_2 + m_2 l_1 l_{c2} sen\theta_2 {\dot{\theta}_1}^2 + m_2 g l_{c2} cos(\theta_1 + \theta_2)$$

(2.6)

2.1.2.1 Representación en variables de estado.

Introduciendo las siguientes constantes en las ecuaciones anteriores

$$\mu_1 = m_1 {l_{c1}}^2 + m_2 {l_1}^2 + I_{zz1}$$
$$\mu_2 = m_2 {l_{c2}}^2 + I_{zz2}$$
$$\mu_3 = m_2 l_1 l_{c2}$$
$$\mu_4 = m_1 l_{c1} + m_2 l_1$$
$$\mu_5 = m_2 l_{c2}$$
$$p = \mu_1 \mu_2 - {\mu_3}^2 cos^2\theta_2$$

Y desacoplando las dos variables $\ddot{\theta}_1$ y $\ddot{\theta}_2$ se tiene:

$$\ddot{\theta}_1 = \frac{1}{p}\left[-\mu_2 \tau + \mu_2 \mu_3 sen\theta_2 (\dot{\theta}_1 + \dot{\theta}_2)^2 + 0{,}5{\mu_3}^2 sen(2\theta_2){\dot{\theta}_1}^2 - \mu_2 \mu_4 g cos\theta_1 (\mu_2 + \mu_3 cos\theta_2) - {\mu_3}^2 sen(2\theta_2)\left({\dot{\theta}_1}^2 + \dot{\theta}_1\dot{\theta}_2 + 0{,}5{\dot{\theta}_2}^2\right) - \mu_5 g cos(\theta_1 + \theta_2)(\mu_1 + \mu_3 cos\theta_2)\right]$$

(2.7)

$$\ddot{\theta}_2 = \frac{1}{p}\left[-(\mu_2 + \mu_3 cos\theta_2)\tau - \mu_3 sen\theta_2\left(\mu_1{\dot{\theta}_1}^2 + \mu_2(\dot{\theta}_1 + \dot{\theta}_2)^2\right) + \mu_4 g cos\theta_1(\mu_2 + \mu_3 cos\theta_2) - {\mu_3}^2 sen(2\theta_2)\left({\dot{\theta}_1}^2 + \dot{\theta}_1\dot{\theta}_2 + 0{,}5{\dot{\theta}_2}^2\right) - \mu_5 g cos(\theta_1 + \theta_2)(\mu_1 + \mu_3 cos\theta_2)\right]$$

(2.8)

Con lo cual el modelo dinámico del Pendubot sin fricción puede representarse en variables de estado al hacer la siguiente asignación de las variables de estados:

$$x_1 = \theta_1$$
$$x_2 = \theta_2$$

$$x_3 = \dot{\theta}_1$$
$$x_4 = \dot{\theta}_2$$
$$\mathbf{u} = \tau$$

Con esta asignación la dinámica del sistema Pendubot está dada por:

$$\dot{X}(t) = F(x) + G(x)u(t)$$
$$y(t) = h(x)$$

Donde:

$$F(x) = \begin{bmatrix} x_3 \\ x_4 \\ f_1(x) \\ f_2(x) \end{bmatrix} \qquad G(x) = \begin{bmatrix} 0 \\ 0 \\ g_1(x) \\ g_2(x) \end{bmatrix}$$

$$h(x) = [x_1 \quad x_2]$$

$f_1(x)$, $f_2(x)$, $g_1(x)$ y $g_2(x)$ están denotadas en (2.7) y (2.8) respectivamente.

2.1.2.2 Puntos de equilibrio

Las condiciones de equilibrio se determinan a partir de las siguientes ecuaciones:

$$\begin{aligned}\tau_1 = {} & (m_1 l_{c1}{}^2 + m_2 l_1{}^2 + 2m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}{}^2 + I_{zz1} + I_{zz2})\ddot{\theta}_1 \\ & - 2m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_1 \dot{\theta}_2 - m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_2{}^2 \\ & + (m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}{}^2 + I_{zz2})\ddot{\theta}_2 + m_1 g l_{c1} cos\theta_1 \\ & + m_2 g l_{c2} cos(\theta_1 + \theta_2) + m_2 g l_1 cos\theta_1\end{aligned}$$

$$\begin{aligned}0 = {} & (m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}{}^2 + I_{zz2})\ddot{\theta}_1 + (m_2 l_{c2}{}^2 + I_{zz2})\ddot{\theta}_2 + m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_1{}^2 \\ & + m_2 g l_{c2} cos(\theta_1 + \theta_2)\end{aligned}$$

Los puntos de equilibrio son muy importantes debido a que cada una de estos puntos requiere una condición de control diferente, esto es que los valores del controlador varían.

Cuando el Pendubot es controlado en una configuración de equilibrio, la velocidad y aceleración de cada eslabón son nulas, esto es:

$$\tau_1 = (m_1 l_{c1} + m_2 l_1) g cos\theta_1 + m_2 g l_{c2} cos(\theta_1 + \theta_2) \quad (2.9)$$

$$0 = m_2 g l_{c2} cos(\theta_1 + \theta_2) \quad (2.10)$$

Los puntos de equilibrio del sistema se determinan a partir de las ecuaciones anteriores, para que la ecuación (2.10) se cumpla se debe tener que $cos(\theta_1 + \theta_2)$ sea cero, con esto se tiene:

$$\theta_1 + \theta_2 = \frac{\pi}{2} \quad (2.11)$$

Introduciendo la expresión anterior (2.11) en (2.9), la señal de control requerida está dada por la siguiente ecuación:

$$\tau_1 = (m_1 l_{c1} + m_2 l_1) g cos\theta_1 = \mu_4 g cos\theta_1 \quad (2.12)$$

La señal de control en función de las ecuaciones paramétricas se encuentran en unidades de voltaje [3].

2.1.3 APROXIMACIÓN LINEAL DEL MODELO MATEMÁTICO

La aproximación lineal se hace en los puntos de equilibrio ya que corresponde a aquellos en los cuales se quiere regular el sistema.

Para la aproximación lineal se empleará la expansión de las ecuaciones en series de Taylor, esto es diferenciando la ecuaciones (2.7) y (2.8) respecto a las variables de estado, obteniéndose las matrices A y B.

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & 0 & 0 \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \end{bmatrix}$$

En la sección anterior se obtuvieron las ecuaciones (2.11) y (2.12) las cuales permiten estudiar los puntos de equilibrio del Pendubot.

Para este proyecto los puntos de interés son las posiciones tope $\left(\frac{\pi}{2}, 0\right)$ y media $\left(-\frac{\pi}{2}, \pi\right)$, se evalúa en los puntos de equilibrio cada Matriz, y se obtienen los siguientes resultados:

- Posición media, $\dot{x} = Ax + Bu$:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -63{,}9961 & 62{,}2213 & 0 & 0 \\ 64{,}6366 & 27{,}7913 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 129{,}5721 \\ -37{,}8246 \end{bmatrix}$$

- Posición tope, $\dot{x} = Ax + Bu$:

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 63{,}9961 & -62{,}2213 & 0 & 0 \\ -63{,}3556 & 152{,}2339 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 129{,}5721 \\ -221{,}3196 \end{bmatrix}$$

Estas matrices constituyen el modelo lineal del sistema en las configuraciones tope y media del Pendubot. Como se puede observar cada punto de equilibrio tiene una linealización diferente, por lo que cada punto requiere que el control tenga diferentes valores de ganancia.

El objetivo del controlador es el de llevar los dos péndulos de la posición de equilibrio estable a una posición de equilibrio inestable en donde el sistema se balancea. La estrategia de control se divide en dos partes. La primera de ellas emplea un controlador que se encarga de llevar las dos barras cerca de la posición deseada mediante la oscilación del eslabón uno [2] y se desarrolla aplicando el método de Linealización Parcial por Realimentación [5], la segunda estrategia mantiene estable el sistema en la posición de equilibrio empleando una técnica lineal para la realimentación óptima del estado del sistema.

Cada una de las estrategias se define en el siguiente capítulo.

CAPÍTULO 3

DISEÑO DE LOS CONTROLADORES DEL SISTEMA PENDUBOT

Existen sistemas que son fáciles de controlar debido a que poseen un número adecuado de actuadores, tales como robots, automóviles, máquinas de producción, sin embargo existen algunos sistemas que son difíciles de controlar debido al número limitado de actuadores, es decir tienen menor cantidad de actuadores que grados de libertad (sistemas subactuados) y surgen de varias formas, por diseño, por naturaleza propia o debido a la aplicación; como son aviones, helicópteros o satélites.

El Pendubot, tal y como ya se explicó en el capítulo 1, es un sistema electromecánico subactuado que consiste de dos eslabones rígidos interconectados. La primera unión está actuada por un motor de DC y la segunda se encuentra sin actuador (figura 1.1).

El problema es encontrar una ley de control que permita llevar a los eslabones desde la posición estable de equilibrio a las posiciones de equilibrio inestables del Pendubot y mantenerlo en esa posición, para ello se utilizan dos técnicas de control, descritas en este capítulo.

3.1 DISEÑO DEL CONTROL

El control del Pendubot está dividido en dos problemas principales, Balanceo (Swing-up) y Equilibrio.

Estos dos problemas son resueltos con diferentes controladores.

3.1.1 CONTROL DE EQUILIBRIO

Para estabilizar el Pendubot en las posiciones de equilibrio, se usa un Regulador Cuadrático Lineal Optimo (LQR, Linear Quadratic Regulator), ya que este controlador da un control óptimo de estabilización.

Los parámetros del controlador se derivan usando el sistema linealizado evaluado alrededor del punto de equilibrio. Obteniéndose una realimentación de torque como:

$$\tau = \tau_0 - K(x - x_0)$$

Donde *K* es la matriz de ganancia óptima de 1x4.

K se calcula reduciendo al mínimo el índice de desempeño cuadrático, *J*, dado por la ecuación:

$$J\min = \int \Delta x^T Q \Delta x + \Delta u^T R \Delta u$$

Esto se realiza mediante la solución de la ecuación matricial de Riccati:

$$A^T P + PA + Q = PBR^{-1}B^T P$$

Entonces *K* está dada por la ecuación:

$$K = R^{-1}B^T P$$

El esquema de control resultante se muestra en la figura 3.1

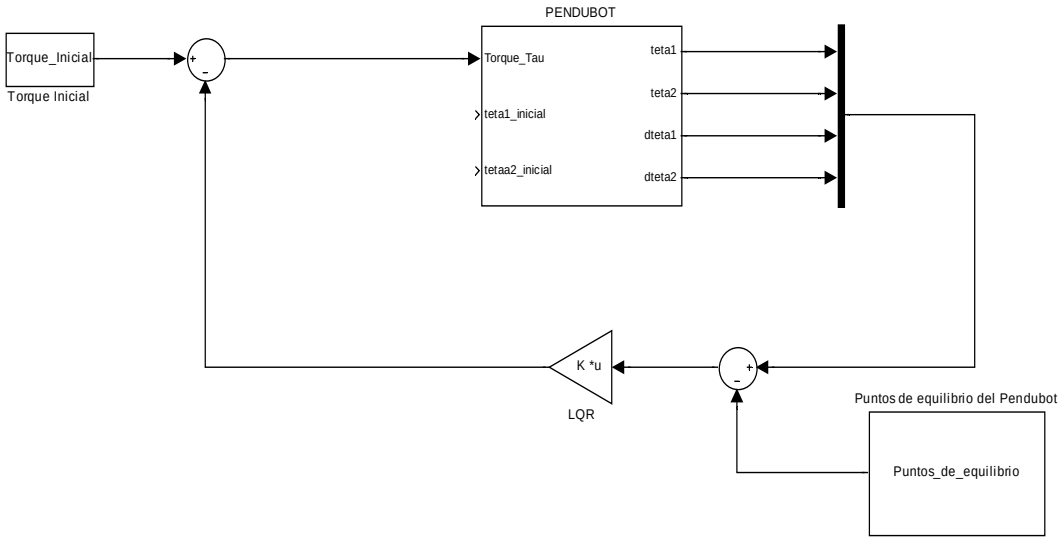


Figura 3.1 Esquema de control del sistema

Para obtener los valores del vector *K* de ganancias del controlador por realimentación de estados que estabilice el sistema, se utiliza la instrucción de MATLAB "*lqrd*", el cual resuelve la ecuación de Riccati, además se puede utilizar el archivo *lqrdpen.m* dadas en [3]. Para no usar otro programa adicional, se realiza un subvi para determinar los valores de *K* (se detalla en el capítulo 5).

Las matrices utilizadas en la función de MATLAB son *A* y *B* que se obtienen del proceso de linealización del sistema alrededor del punto de equilibrio, y las matrices Q y R se determinan de la siguiente manera:

- Los elementos de la matriz Q se seleccionan para ponderar la importancia relativa de los diferentes componentes del vector de estado. Los estados de interés corresponden a la posición de los enlaces, es decir, a los estados x_1 y x_2. La matriz Q empleada inicialmente es:

  - Posición Tope

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

  - Posición Media

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- La matriz R se selecciona para ponderar el gasto de energía en la acción de control, es decir, mientras mayor sea la ponderación asignada, la magnitud de la señal de control será menor. La matriz R empleada inicialmente es:

  - Posición Tope

$$R = [1,1]$$

  - Posición Media

$$R = [1,1]$$

El vector obtenido de *K* para cada posición es:

- Posición Tope

$$K = [-10{,}7180 \quad -10{,}1647 \quad -2{,}1136 \quad -1{,}4222]$$

- Posición Media

$$K = [4.8134 \quad 7.3641 \quad 0.495 \quad 1.0054]$$

3.1.2 CONTROL DE BALANCEO

El control de balanceo consiste en llevar los eslabones desde su posición de equilibrio estable a las posiciones inestables, a través de los movimientos del primer eslabón, para este problema se utilizará las técnicas de linealización parcial [2]. El control de Balanceo se diseñará en dos pasos:

Linealización Parcial.

La linealización parcial permite transformar, de forma parcial o total, la dinámica no lineal del sistema a controlar en una dinámica lineal. La idea principal del método de linealización parcial por retroalimentación de estados es para tomar en cuenta el efecto no lineal del segundo eslabón sobre el primero.

Compensador PD.

Se usará un compensador PD sobre el sistema parcialmente linealizado, para llevar al primer eslabón a su posición tope o media a través de una trayectoria fija o señal de referencia, donde el segundo eslabón se moverá libremente sin ningún control.

Luego se empleará el esquema de prealimentación para asegurar que el segundo eslabón oscile de forma controlada (indirectamente, a través del primer eslabón), hasta acercarse al punto de equilibrio superior y luego conmutar a la ley de control de equilibrio.

3.1.2.1 Linealización Parcial

Dado que el segundo eslabón es no-actuado, esta parte de la dinámica no se puede linealizar. Sin embargo linealizar uno de los grados de libertad, facilita el

diseño de un lazo de control externo que seguirá una trayectoria dada para el grado de libertad linealizado.

La idea es alimentar el sistema con un par τ modificado con el fin de cancelar la no linealidad.

Esto se hace a través de un lazo interno dentro del sistema y dar una nueva entrada de control, llamada v_1 (ver Figura 3.2).

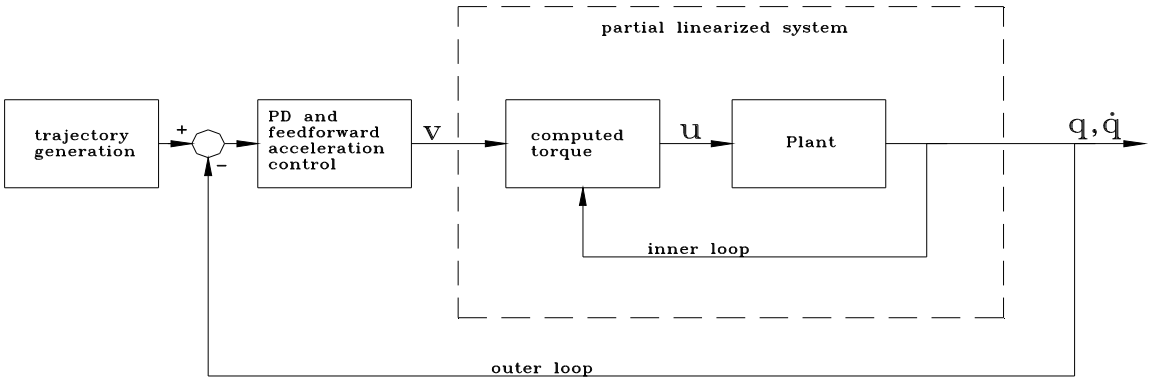


Figura 3.2 Diagrama de bloques del control por linealización parcial [11]

Los siguientes pasos muestran como linealizar parcialmente el sistema, entre la entrada de control τ y $\ddot{\theta}_1$

Las ecuaciones de movimiento del sistema dadas por (2.5) y (2.6), se escriben nuevamente:

$$\begin{aligned}\tau_1 = {} & (m_1 l_{c1}^2 + m_2 l_1^2 + 2m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}^2 + I_{zz1} + I_{zz2})\ddot{\theta}_1 \\ & - 2m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_1 \dot{\theta}_2 - m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_2^2 \\ & + (m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}^2 + I_{zz2})\ddot{\theta}_2 + m_1 g l_{c1} cos\theta_1 \\ & + m_2 g l_{c2} cos(\theta_1 + \theta_2) + m_2 g l_1 cos\theta_1\end{aligned} \tag{3.1}$$

$$\begin{aligned}0 = {} & (m_2 l_1 l_{c2} cos\theta_2 + m_2 l_{c2}^2 + I_{zz2})\ddot{\theta}_1 + (m_2 l_{c2}^2 + I_{zz2})\ddot{\theta}_2 + m_2 l_1 l_{c2} sen\theta_2 \dot{\theta}_1^2 \\ & + m_2 g l_{c2} cos(\theta_1 + \theta_2)\end{aligned} \tag{3.2}$$

Despejando en la ecuación (3.2) la aceleración angular del eslabón dos, $\ddot{\theta}_2$, y sustituyendo en (3.1), se obtiene una nueva ecuación para el torque, y utilizando las constantes introducidas en el capítulo anterior, se tiene:

$$\tau_1 = \frac{1}{u_2}\left[p\ddot{\theta}_1 - u_2u_3 sen\theta_2(\dot{\theta}_1 + \dot{\theta}_2)^2 - 0{,}5u_3{}^2 sen(2\theta_2)\dot{\theta}_1 + u_2u_4 g cos\theta_1 - u_3u_5 g cos\theta_2 cos(\theta_1 + \theta_2)\right] \quad (3.3)$$

Usando la siguiente notación:

$$\propto(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = \frac{1}{u_2}\left(-u_2u_3 sen\theta_2(\dot{\theta}_1 + \dot{\theta}_2)^2 - 0{,}5u_3{}^2 sen(2\theta_2)\dot{\theta}_1 + u_2u_4 g cos\theta_1 - u_3u_5 g cos\theta_2 cos(\theta_1 + \theta_2)\right)$$

$$\beta(\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2) = \frac{1}{u_2}p$$

Se define una nueva variable para suprimir los términos no lineales:

$$v_1 = \ddot{\theta}_1 \quad (3.4)$$

El lazo de control interno que linealiza la dinámica de θ_1, se puede definir ahora como:

$$\tau_1 = v_1\beta(\theta_i) + \propto(\theta_i) \quad (3.5)$$

Utilizando esta realimentación para la linealización se describe un nuevo sistema dinámico ya linealizado, con las siguientes ecuaciones.

$$v_1 = \ddot{\theta}_1 \quad (3.6)$$

$$\ddot{\theta}_2 = -\frac{1}{u_2}\left[(u_3 cos\theta_2 + u_2)v_1 + u_3 sen\theta_2\dot{\theta}_1 + u_5 g cos(\theta_1 + \theta_2)\right] \quad (3.7)$$

3.1.2.2 Controlador PD

El sistema dado por las ecuaciones (3.6) y (3.7) es ahora lineal respecto a $\ddot{\theta}_1$, donde $\dot{\theta}_1$ es la salida del sistema y v_1 es la nueva variable manipulada a partir de la cual se obtiene el torque.

En base a la nueva dinámica del sistema dada por (3.6) y (3.7), se diseña el controlador PD con pre-alimentación de la aceleración [5], como sigue:

$$v_1 = \ddot{\theta}_1^d + k_d\left(\dot{\theta}_1^d - \dot{\theta}_1\right) + k_p\left(\theta_1^d - \theta_1\right) \qquad (3.8)$$

Donde $\ddot{\theta}_1^d, \dot{\theta}_1^d$ y θ_1^d son la aceleración, velocidad y posición deseada o de referencia respectivamente. k_p y k_d son las ganancias positivas que se deben sintonizar para la generación de la trayectoria.

3.1.2.2.1 Balanceo Tope

Gracias a la realimentación de la linealización parcial, es posible conseguir la posición angular de θ_1, con una señal de referencia estática haciendo oscilar el sistema desde la posición estable de equilibrio a su posición tope.

Por lo tanto para lograr llegar a la posición tope se hace ajustando la señal de referencia o trayectoria en $\frac{\pi}{2}$. Una vez llegado a esta posición se debe conmutar al control de equilibrio para poder mantenerlo en esta posición.

Los valores de las constates k_p y k_d han sido afinadas en orden para llegar a la posición tope, con una velocidad angular lo más pequeña posible para el eslabón dos, θ_2.

La parte proporcional tiene que ser afinada para conseguir la máxima respuesta del sistema hasta llegar a la saturación. La parte derivativa se afina para tener un movimiento suave del eslabón dos.

Para el Pendubot se ha llegado a conseguir los siguientes valores para los parámetros del controlador PD, en la posición tope.

$$k_p = 1{,}0282$$
$$k_d = 0{,}0527$$
$$\theta_1^d = \frac{\pi}{2}$$

3.1.2.2.2 Balanceo Medio

El balanceo para la posición media es más complicado que la anterior. Una señal de referencia estática no es suficiente para llevar los eslabones a las pociones

deseadas. La meta del balanceo medio es llevar al eslabón dos por una trayectoria adecuada hasta la posición deseada y luego conmutar al controlador de equilibrio.

Para conseguir esto se debe almacenar energía para llevar el eslabón dos a la posición deseada.

Por lo que adicionar una señal ayudará al sistema a conseguir esta posición. Esto es que la señal de referencia inicial debe ser una señal contraria al movimiento para inyectar energía al sistema lo suficiente como para que se alcance la posición deseada.

Luego se afinan los parámetros del controlador PD igual que el caso anterior, variándolos poco a poco hasta que llegue a la posición deseada. Finalmente los valores que se obtuvo fueron:

$$k_p = 0{,}9457$$

$$k_d = 0{,}0359$$

$$\theta_1^d = -\frac{\pi}{2}$$

La señal inicial de referencia aplicada es de -0,94 V durante 0,5948 segundos, como se muestra en la siguiente figura.

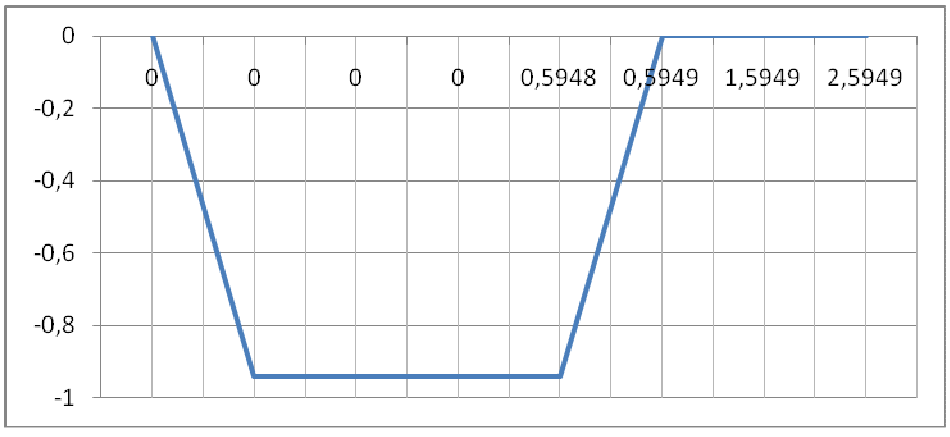
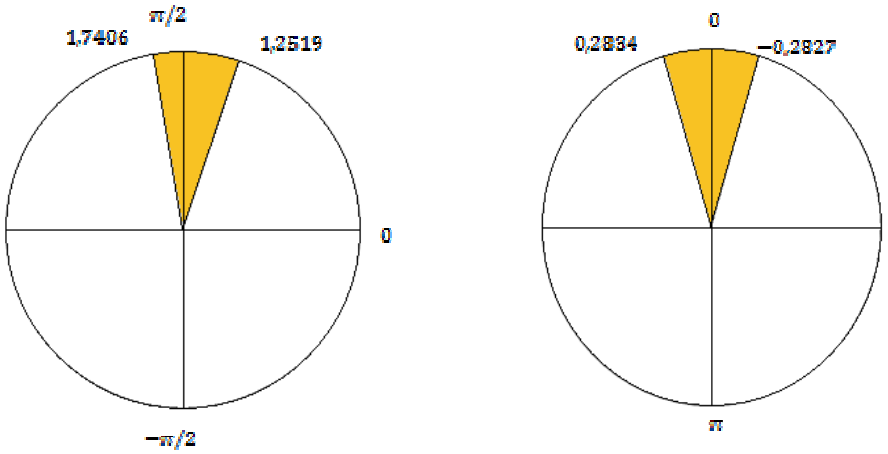


Figura 3.3 Señal de referencia aplicada al Pendubot para el balanceo medio

3.2 SISTEMA DE CONMUTACIÓN

Todos los controladores antes mencionados tienen su respectiva función, los de balanceo llevar el Pendubot alrededor de los puntos de equilibrio inestables y los controladores de equilibrio mantener al Pendubot en equilibrio.

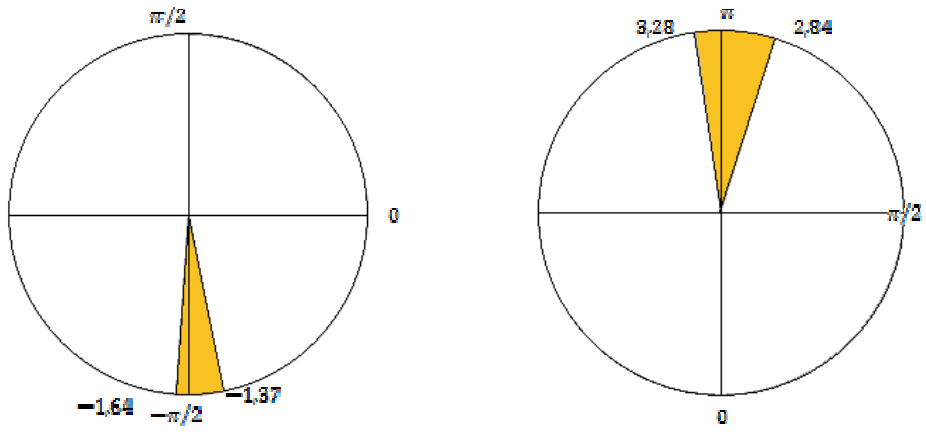
La figura 3.5 muestra los limites de conmutación desde el control de balanceo al control de equilibrio para la posición tope.



(a) Eslabón 1 **(b)** Eslabón 2

Figura 3.5 Límites de conmutación del Pendubot Balanceo Tope

La figura 3.6 muestra los limites de conmutación desde el control de balanceo al control de equilibrio para la posición media.



(a) Eslabón 1 (b) Eslabón 2

Figura 3.6 Límites de conmutación del Pendubot Balanceo Medio.

Con el fin de que el sistema pueda ser controlado por los dos algoritmos diseñados, estos deben actuar en determinados momentos según sean las posiciones angulares de los eslabones, como se mostró en los límites de conmutación.

Una vez que se tiene los controladores y sus límites de conmutación, se implementa en la FPGA, en este dispositivo se puede programar toda la lógica que se necesita, además de la interfaz para el usuario. Se diseña el control con toda la información del funcionamiento del sistema y para programar la FPGA, se utiliza el entorno gráfico de LAVIEW.

CAPÍTULO 4

DESCRIPCIÓN DEL FPGA

En la última década se han desarrollado diferentes diseños de hardware orientados a control de sistemas subactuados. Por un lado están los que se basan en microprocesadores de propósito general adecuadamente programados. Por otro lado están los que utilizan un hardware totalmente específico. Ambos extremos tienen sus ventajas e inconvenientes.

En la actualidad existen dispositivos que contienen lo mejor de los dispositivos mencionados anteriormente, como son las FPGA's.

Una **FPGA** (*Field Programmable Gate Array*) es un dispositivo semiconductor que contiene bloques de lógica cuya interconexión y funcionalidad se puede programar. La lógica programable puede reproducir desde funciones tan sencillas como las llevadas a cabo por una puerta lógica o un sistema combinacional hasta complejos sistemas en un chip. [1]

Las FPGA's tienen la ventaja de ser reprogramables (lo que aumenta una enorme flexibilidad al flujo de diseño), los circuitos se "ejecutan" más rápido que en otros dispositivos ya que su ejecución es en paralelo, por lo que los circuitos no necesitan competir por los mismos recursos. Cada tarea de procesos se asigna a una sección dedicada del dispositivo y puede ejecutarse de manera autónoma sin ser afectada por otros bloques de lógica. Como resultado, el rendimiento de una parte de la aplicación no se ve afectado cuando se agregan otros procesos. [12]

Además sus costes de desarrollo y adquisición son mucho menor para pequeñas cantidades de dispositivos y el tiempo de desarrollo es también menor.

Las FPGA son el resultado de la convergencia de dos tecnologías diferentes, los dispositivos lógicos programables (PLDs [Programmable Logic Devices]) y los circuitos integrados de aplicación específica (ASIC [application-specific integrated circuit]).[1]

4.1 EVOLUCIÓN DE LOS DISPOSITIVOS PROGRAMABLES.

Se entiende por dispositivo programable aquel circuito de propósito general que posee una estructura interna que puede ser modificada por el usuario final.

El primer dispositivo que cumplió estas características fue la memoria PROM, luego los PLD's los cuales son una matriz de puertas AND conectadas a otra matriz de puertas OR más biestables, mas tarde aparecieron las PLA (Programmable Logic Array), estos dispositivos son muy simples y producen buenos resultados con funcionalidades sencillas (sólo combinacional).

En la actualidad las FPGA's, introducidas por Xilinx en 1984, son dispositivos programables por el usuario, las cuales consisten en una matriz bidimensional de bloques configurables que se pueden conectar mediante recursos generales de interconexión. Estos recursos incluyen segmentos de pista de diferentes longitudes, más unos conmutadores programables para enlazar bloques a pistas o pistas entre sí. En realidad, lo que se programa en una FPGA son los conmutadores que sirven para realizar las conexiones entre los diferentes bloques, más la configuración de los bloques. [13]

4.2 ARQUITECTURA DE LA FPGA SPARTAN 3 DE XILINX

Las FPGA Spartan 3 de Xilinx están conformadas por un conjunto de Bloques Lógicos Configurables (Configurable Logic Blocks: CLBs) rodeados por un perímetro de Bloques Programables de entrada/salida (Programmable Input/Output Blocks: IOBs). Estos elementos funcionales están interconectados por una jerarquía de canales de conexión (Routing Channels), la que incluye una red de baja capacitancia para la distribución de señales de reloj de alta frecuencia.

Los cinco elementos funcionales programables que la componen son los siguientes:

Bloques de entrada/salida (Input/Output Blocks – IOBs): Controlan el flujo de datos entre los pines de entrada/salida y la lógica interna del dispositivo. Soportan

flujo bidireccional más operación tri-estado y un conjunto de estándares de voltaje e impedancia controlados de manera digital.

Bloques Lógicos configurables (Configurable Logic Blocks – CLBs): Contienen Look-Up Tables basadas en tecnología RAM (LUTs) para implementar funciones lógicas y elementos de almacenamiento que pueden ser usados como flip-flops o como latches.

Bloques de memoria RAM (Block RAM): Proveen almacenamiento de datos en bloques de 18 Kbits con dos puertos independientes cada uno.

Bloques de multiplicación que aceptan dos números binarios de 18 bit como entrada y entregan uno de 36 bits.

Administradores digitales de reloj (Digital Clock Managers – DCMs): Estos elementos proveen funciones digitales auto calibradas, las que se encargan de distribuir, retrasar arbitrariamente en pocos grados, desfasar en 90, 180, y 270 grados, dividir y multiplicar las señales de reloj de todo el circuito.

Los elementos descritos están organizados como se muestra en la figura 4.1. Un anillo de IOBs rodea un arreglo regular de CLBs. Atraviesa este arreglo una columna de Bloques de memoria RAM, compuesta por varios bloques de 18 Kbit, cada uno de los cuales está asociado con un multiplicador dedicado. Los DCMs están colocados en los extremos de dichas columnas.

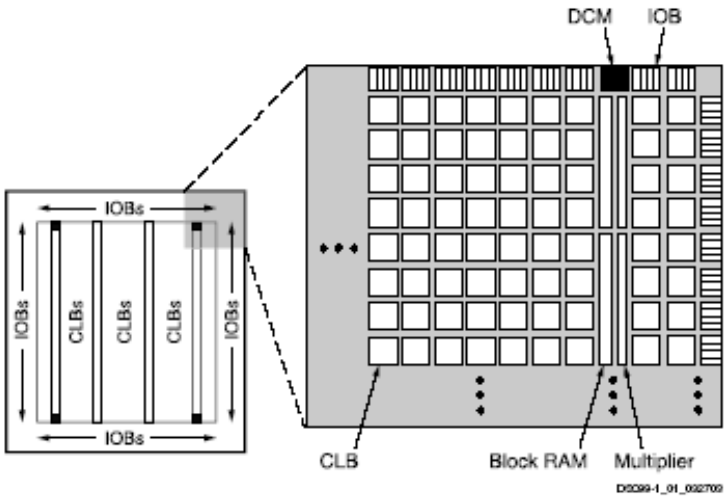


Figura 4.1 Arquitectura de la Spartan 3.

Para tener una descripción más detallada de cada uno de los elementos funcionales de la FPGA, referirse al Anexo C.

4.3 CARACTERÍSTICAS Y COMPONENTES PRINCIPALES DE LA TARJETA DE DESARROLLO SPARTAN-3E STARTER KIT [9]

La Tarjeta de desarrollo Spartan-3E Starter Kit contiene varios elementos que facilitan mucho el diseño de proyectos [9].

- Características Principales:
  - Xilinx Devices:
    - Spartan-3E FPGA (XC3S500E-4FG320C)
    - CoolRunner™-II CPLD (XC2C64A-5VQ44C)
    - Platform Flash (XCF04S-VO20C)
  - Clocks: 50 MHz crystal clock oscillator
  - Memory:
    - 128 Mbit Parallel Flash
    - 16 Mbit SPI Flash
    - 64 MByte DDR SDRAM
- Conectores e Interfaces:
  - Ethernet 10/100 Phy
  - JTAG USB download
  - Two 9-pin RS-232 serial port
  - PS/2- style mouse/keyboard port.
  - Rotary encoder with push button
  - Four slide switches
  - Eight individual LED outputs
  - Four momentary-contact push buttons
  - 100-Pin expansion connection ports
  - Three 6-pin expansion connectors
  - Display: 16 character - 2 Line LCD

La siguiente figura muestra el diagrama de bloques de la tarjeta Spartan-3E.

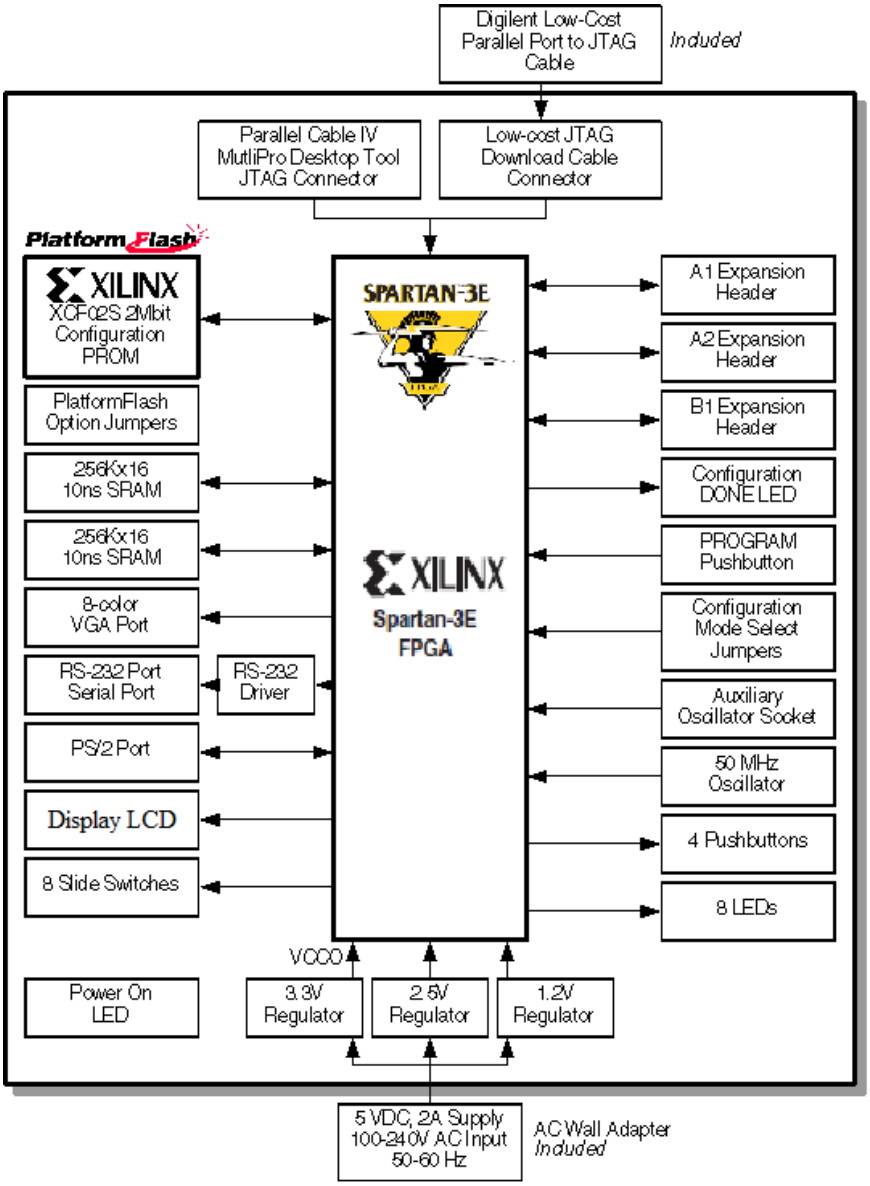


Figura 4.2 Diagrama de Bloques de la Spartan-3E.

Los pines de Entrada/Salida de los puertos solo pueden recibir por defecto +3.3V como entrada y entregar +3.3V como salida, esto es en estándar LVTTL (Baja Tensión TTL). Sin embargo se puede cambiar el voltaje a 2.5 V, usando unos jumpers en el módulo, este voltaje es usado en algunas E/S del FPGA. Cualquier voltaje mayor a estos pueden dañar los puertos del módulo e inclusive al mismo.

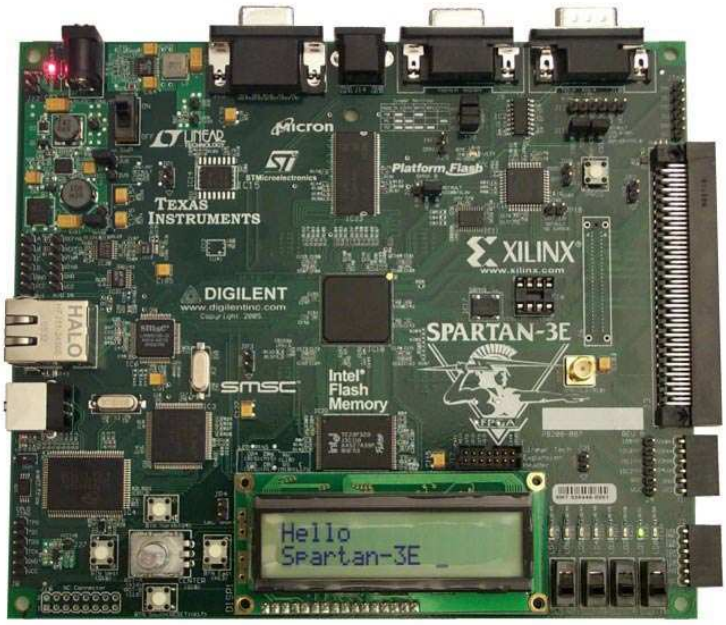


Figura 4.3 Módulo FPGA Spartan-3E

Se dará una breve explicación de los componentes utilizados en este proyecto, para más detalle de la tarjeta referirse a [9].

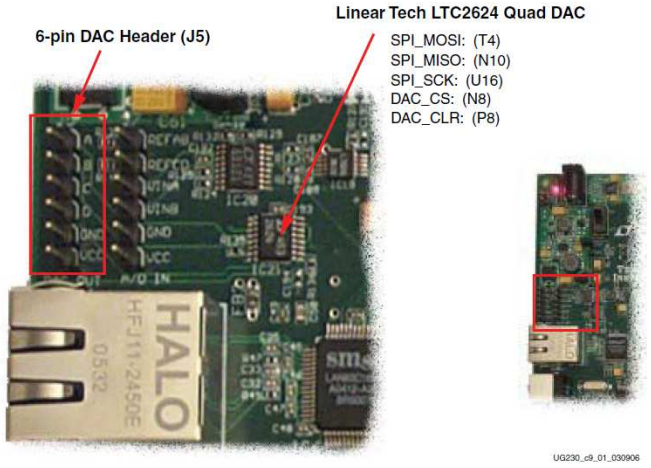
4.3.1 CONVERSOR DIGITAL-ANÁLOGO (DAC)

El FPGA Spartan-3E Starter Kit incluye 4 canales DAC con resolución 12 bits, para comunicarse con cada canal, se usa comunicación SPI (Serial Peripheral Interface).

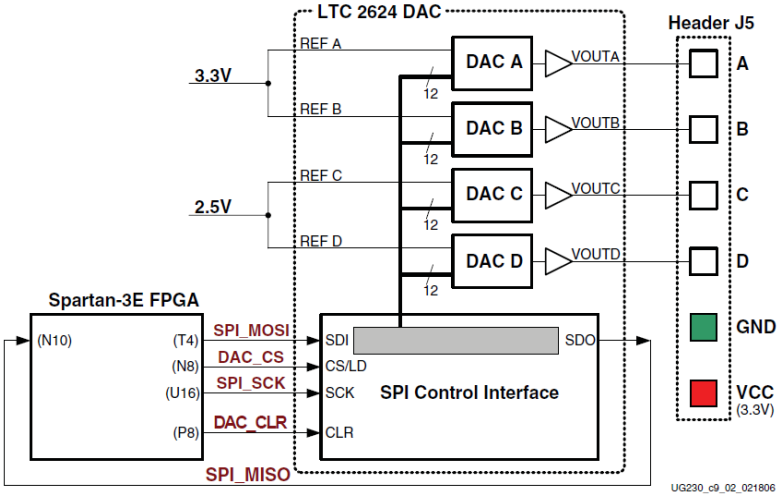
En la figura 4.4 (a) se muestra la localización del conversor y en la figura 4.4 (b) se muestra la conexión esquemática del DAC con la FPGA Spartan-3E.

En la tabla 4.1 se muestra la lista de señales que se deben configurar entre el FPGA y el DAC

Como el bus SPI es compartido con otros dispositivos del módulo, es importante que estos se deshabiliten cuando se utilice la comunicación entre el FPGA y el DAC.
La tabla 4.2 muestra la lógica y las señales para deshabilitar estos dispositivos.



(a)



(b)

Figura 4.4 (a) Ubicación Conversor Digital-Análogo [9]

(b) Conexión Esquemática del DAC con la Spartan-3E [9]

Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA→DAC	Serial data: Master Output, Slave Input
DAC_CS	N8	FPGA→DAC	Active-Low chip-select. Digital-to-analog conversion starts when signal returns High.
SPI_SCK	U16	FPGA→DAC	Clock
DAC_CLR	P8	FPGA→DAC	Asynchronous, active-Low reset input
SPI_MISO	N10	FPGA←DAC	Serial data: Master Input, Slave Output

Tabla 4.1 Señales de Configuración del DAC [9]

Signal	Disabled Device	Disable Value
SPI_SS_B	SPI serial Flash	1
AMP_CS	Programmable pre-amplifier	1
AD_CONV	Analog-to-Digital Converter (ADC)	0
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	1

Tabla 4.2 Lógica para deshabilitar los dispositivos compartidos con el bus SPI [9]

El voltaje especificado para la salida se describe con la siguiente ecuación:

$$V_{out} = \frac{D_{[11:0]}}{4096} x V_{ref}$$

Donde $D_{[11:0]}$ es el valor de 12 bits equivalente al voltaje análogo

El Voltaje de referencia V_{ref} depende de los canales utilizados; esto es, los canales A y B usan un voltaje de referencia de 3.3 V, mientras que los canales C y D usan 2.5 V de referencia.

4.3.2 CONECTORES DE EXPANSIÓN

El módulo Spartan-3E Starter Kit tiene una gran variedad de conectores de expansión, los cuales son:

- Un conector Hirose de 100 pines (J3), los cuales están asociados con 43 E/S del FPGA, incluidos 15 pares de entradas diferenciales.
- 3 conectores de 6 pines cada uno (J1, J2 y J4).
- Conectores de prueba (J6).

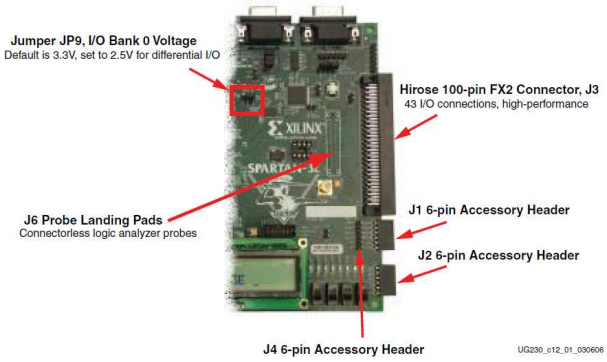


Figura 4.5 Conectores de expansión [9]

En este proyecto se necesitan 4 entradas, se utiliza uno de los 3 conectores de 6 pines, específicamente J1.

La figura 4.6 muestra los 6 pines del conector J1, donde tiene 4 señales que son compartidas con el conector J3 (Hirose 100-pin), también se puede utilizar como una fuente de 3.3 V.

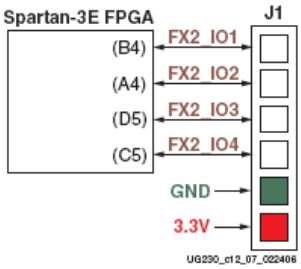


Figura 4.6 Conector J1 [9]

4.4 APLICACIONES DE UNA FPGA

Cualquier circuito de aplicación específica puede ser implementado en un FPGA, siempre y cuando esta disponga de los recursos necesarios. Las aplicaciones donde más comúnmente se utilizan las FPGA incluyen a los DSP (procesamiento digital de señales), radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bioinformática, emulación de hardware de computadora, entre otras. Cabe notar que su uso en otras áreas es cada vez mayor, sobre todo en aquellas aplicaciones que requieren un alto grado de paralelismo.

Esta tarjeta se puede emplear para múltiples aplicaciones. Sin embargo, resulta especialmente útil en los siguientes campos:

Docencia, en el campo de la electrónica y los lenguajes de descripción hardware (HDL), como también los lenguajes de programación gráfica. Por ser una placa libre y estar disponibles todos los esquemas y planos de fabricación, se puede ampliar su funcionalidad, diseñándose placas a su medida (realización de prácticas especiales, proyectos fin de carrera, trabajos de doctorado, etc.).

Conexión con microcontroladores. Puesto que puede funcionar en modo autónomo, resulta muy útil para el desarrollo de periféricos para microcontroladores: controladores de sensores, coprocesadores para hacer ciertas operaciones más rápidamente, etc. Además es posible realizar la carga del bitstream[1] desde el propio sistema microcontrolador, posibilitando el desarrollo de sistemas de hardware reconfigurables.

Robótica. Posibilidad de diseñar CPUs específicas para aplicaciones de robótica así como controladores para periféricos: unidades de PWM para mover servos, temporizadores, controladores para sensores de ultrasonidos, de distancia, controladores modernos, inteligentes, etc.

4.5 PROGRAMACIÓN DE UNA FPGA

La tarea para programar una FPGA primero es definir la función lógica que realizará cada uno de los **CLB**, luego seleccionar el modo de trabajo de cada **IOB** e interconectarlos.

Para ello se tiene entornos de desarrollo especializados en el diseño de sistemas a implementarse en un FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de programación especial. Estos lenguajes de programación especial son conocidos como HDL o *Hardware Description Language* (Lenguajes de Descripción de Hardware). Los HDLs más utilizados son:

[1] Bitstream: Este término es frecuentemente utilizado para describir los datos de configuración que son cargados al FPGA.

1. VHDL
2. Verilog
3. ABEL

En un intento de reducir la complejidad y el tiempo de desarrollo en fases de prototipaje rápido, y para validar un diseño en HDL, existen varias propuestas y niveles de abstracción del diseño. Entre otras, National Instruments con LabVIEW FPGA propone un acercamiento de programación gráfica de alto nivel. [1]

4.5.1 PROGRAMACIÓN GRÁFICA DE ALTO NIVEL.

Tradicionalmente, se ha utilizado la tecnología FPGA con herramientas de programación avanzadas. Sin embargo, como los FPGA's se han vuelto más rápidos y más rentables, en la actualidad se los puede programar con poca o ninguna experiencia en diseño de hardware digital, aprovechando así a las FPGA's para crear soluciones personalizadas. Para abarcar este creciente interés, los proveedores están creando herramientas de más alto nivel que hacen más fácil programar FPGA's y brindar los beneficios de la tecnología FPGA a nuevas aplicaciones.

El Modulo de LABVIEW FPGA de National Instruments extiende las capacidades de desarrollo gráfico de Labview a FPGA's, en donde se puede crear sistemas de medición y control personalizado en hardware sin tener que diseñar en lenguaje descriptor de hardware o a nivel de circuito.

4.5.2 MÓDULO LABVIEW FPGA

LabVIEW (*Laboratory Virtual Instrument Engineering Workbench*) es un lenguaje de programación gráfica para el diseño de sistemas de adquisición de datos, instrumentación y control.

Con LABVIEW FPGA, se puede crear VI's que pueden correr en dispositivos reconfigurables como son las FPGA, no se necesita tener conocimiento de HDL

Lenguaje de descripción de hardware) para diseñar sistemas de hardware personalizado.

El módulo de LABVIEW FPGA es parte de la plataforma LABVIEW Embebida que está compuesta por múltiples componentes de software que permite a los usuarios programar hardware embebido o personalizado con LABVIEW.

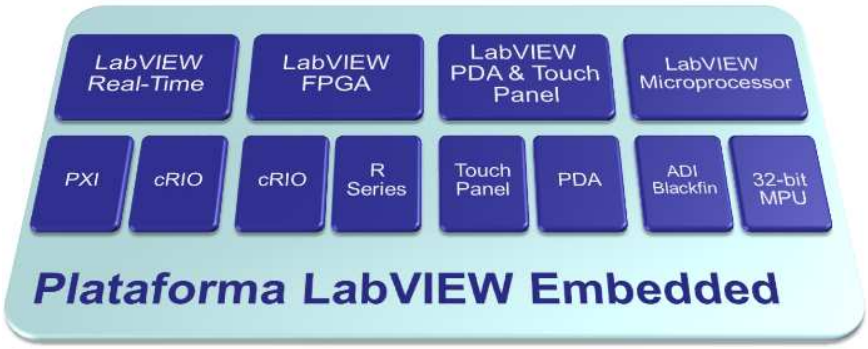


Figura 4.7 Plataforma de LABVIEW Embebida [14]

Esto permite a los usuarios utilizar un ambiente de desarrollo común para trabajar con cualquier hardware embebido, esto es utilizarlo para un sistema de tiempo real o un FPGA en el mismo entorno.

Para lograr esta tecnología National Instruments creó diferentes mecanismos para tomar los diagramas de LABVIEW y pasarlos a través de diferentes compiladores dependiendo del hardware final de ejecución. LABVIEW utiliza diferentes tecnologías para compilar el código de LABVIEW en bitfiles o archivos de bits para descargarlo y ejecutarlo en un chip FPGA.

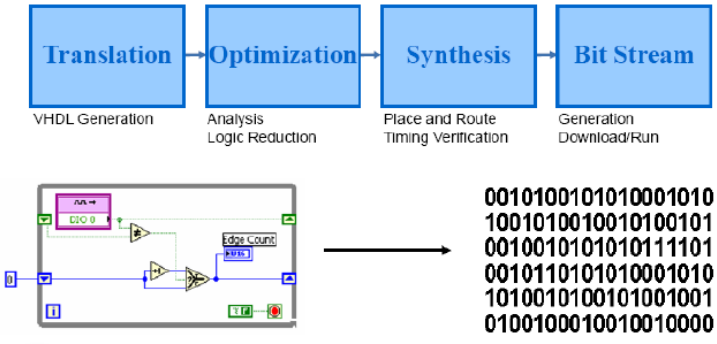


Figura 4.8 Flujo de Compilación de LABVIEW 2010 FPGA [23]

Sistema Embebido: Se entiende por sistema embebido a una combinación de hardware y software de computadora, diseñado para tener una función específica

La lógica de LABIEW es similar a la lógica del FPGA, ya que simplemente se necesita lazos en paralelo para poder realizar funciones en paralelo, como se muestra en la figura 4.9.
El paralelismo es una forma de ejecutar múltiples tareas a la vez.

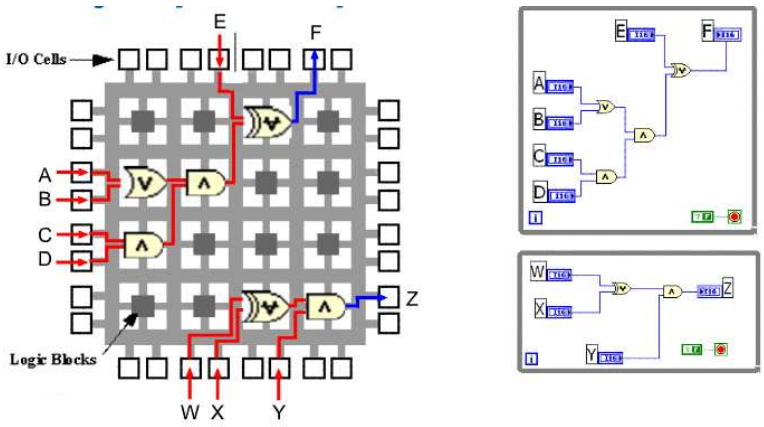


Figura 4.9 Lógica del FPGA y LABVIEW [15]

Un reto en la programación de tareas paralelas es pasar datos a través de múltiples ciclos sin crear una dependencia de dato, para ello se usan diferentes tipos de variables.

4.5.2.1 Entorno LABVIEW FPGA

El entorno de LABVIEW FPGA es similar a un entorno normal de LAVIEW, es decir el VI tiene un panel frontal y un diagrama de bloques, pero la diferencia está en las paletas que contienen opciones limitadas y diferentes a las que se emplean normalmente para crear y modificar los VIs.

4.5.2.1.1 Conceptos Básicos

El *Panel Frontal* de LabVIEW se comunica con el diagrama de bloques de la FPGA para cambiar el estado de los controles e indicadores.

El *Diagrama de Bloques* contiene el código fuente del VI.
Las *Paletas* proporcionan las herramientas que se requiere para crear y modificar tanto el panel frontal como el diagrama de bloques.

Las *variables* en LABVIEW, son elementos del diagrama de bloques que permiten acceder o almacenar datos en otra localización. La localización actual del dato varía dependiendo del tipo de variable:

- Las variables *locales* almacenan datos en controles e indicadores del panel frontal para ser utilizados dentro del mismo VI.
- Las variables *globales* y *compartidas* almacenan datos en localizaciones especiales que se puede acceder desde múltiples VI's.

Las variables permiten saltar el flujo normal de datos al pasar datos desde un lugar a otro sin conectarlos con un cable. Por esta razón, las variables son útiles en las arquitecturas paralelas.

LABVIEW permite encapsular secciones de código común, como son los *SubVI's* para facilitar su reutilización en el diagrama de bloques

Los SubVI's se los puede configurar como subVI's reentrantes y no-reentrantes, esta diferencia es muy importante debido a que un subVI no-reentrante no permite la ejecución en paralelo, sino solo compartir el recurso; mientras que un SubVI's reentrante además de permitir compartir el recurso se puede ejecutar en paralelo, es decir que su ejecución dentro del un mismo diagrama de bloques es múltiple y paralela. Los SubVI's creados en el ambiente FPGA están predeterminados como reentrantes.

Para hacer un subVI no reentrante, se selecciona Ejecución en la Categoría en el menú desplegable del cuadro de diálogo de Propiedades del VI y se quita la marca de la casilla de verificación de ejecución reentrante.

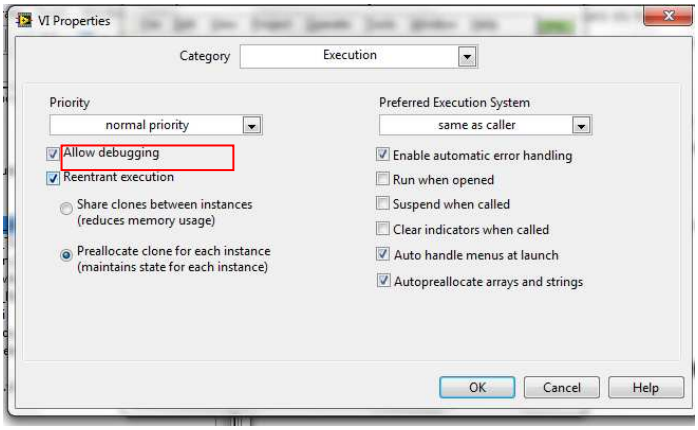


Figura 4.10 Modo de ejecución Reentrante.

La FPGA VI se carga directamente al dispositivo FPGA, mientras que el *host VI* está cargada en la PC. El *host VI* sirve como la interfaz entre la PC y la FPGA.

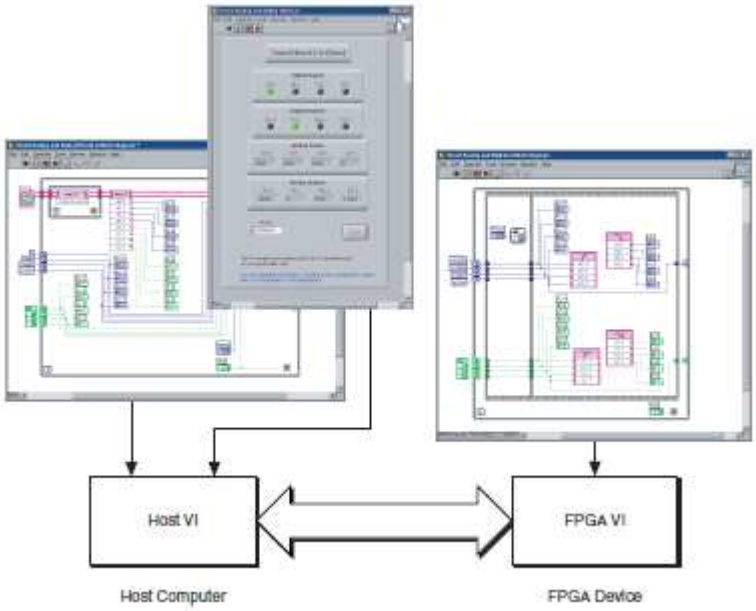


Figura 4.11 Modo de comunicación entre Host y FPGA-VI [24]

4.5.2.1.2 Programación Básica-LABVIEW FPGA

Para crear un proyecto con la Spartan-3E se siguen los siguientes pasos:

1. Se abre un proyecto nuevo *Empty Project.*
2. Se da click-derecho sobre *My Computer* y en este se selecciona el dispositivo: *New» Targets and Devices,* como se muestra en la figura 4.12

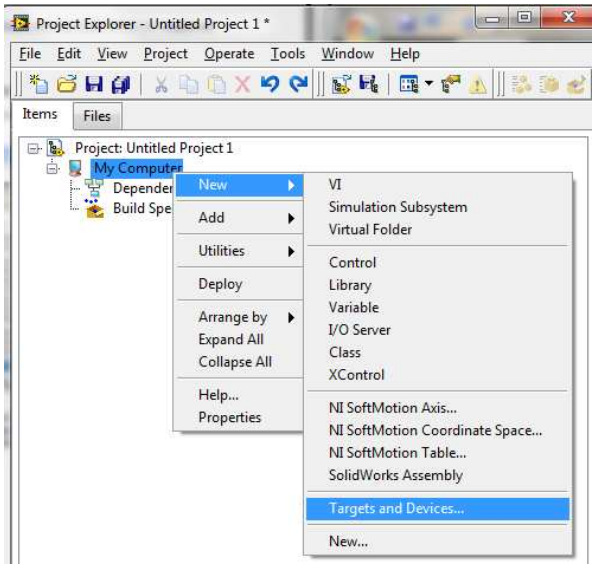


Figura 4.12 Creación de un proyecto Spartan-3E

3. Luego se abre la ventana *Add Targets and Devices* y se seleccina *New target or device*, se expande *Xilinx University Program* y se selecciona *Spartan-3E Starter Board* y se presiona *OK*

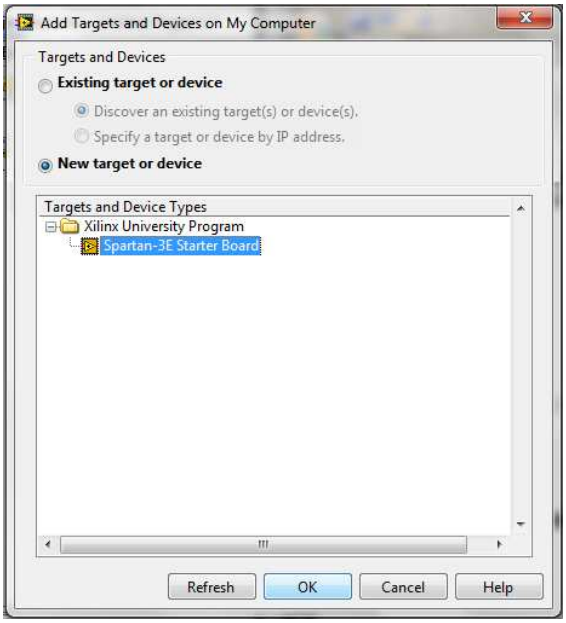


Figura 4.13 Add Spartan-3E Target

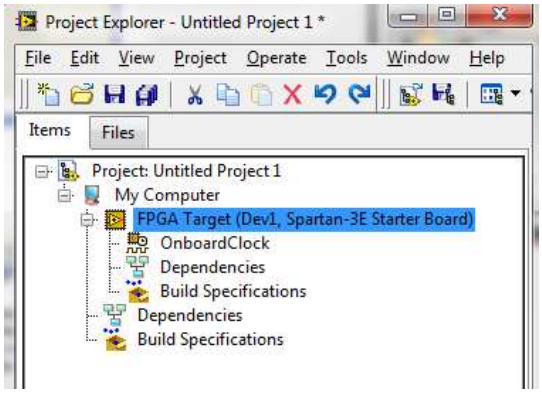


Figura 4.14 Dispositivo agregado al proyecto

4. Ahora que se tiene agregado el dispositivo en el proyecto figura 4.14, se crea una FPGA VI, se da click-derecho sobre el nombre del dispositivo agregado *FPGA Target (Dev1, Spartan-3E Starter Board)* y se selecciona *New» VI* y se tendrá el FPGA VI añadido al proyecto, se debe notar que el FPGA VI está en el árbol del dispositivo FPGA, figura 4.15

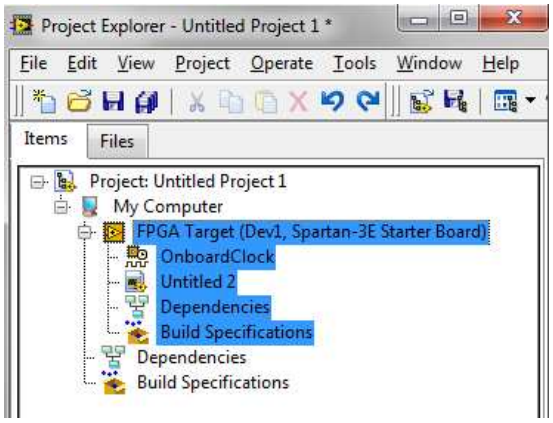


Figura 4.15 FPGA VI añadido al proyecto

Una vez creado el proyecto, se deben agregar las Entradas y/o salidas que se necesiten, para esto se siguen los siguientes pasos:

1. Se da click-derecho en el nombre del dispositivo agregado *FPGA Target (Dev1, Spartan-3E Starter Board)* y se selecciona *New» FPGA I/O* como se muestra en la figura 4.16
2. Ahora se tendrá una nueva ventana, en la cual se encuentran todos las entradas y salidas del dispositivo, además de las variables para configurar

los módulos existentes en la tarjeta como son el conversor digital-análogo, conversor análogo-digital, LCD, etc.

3. Se selecciona el elemento a agregar y se lo añade en la tabla, una vez seleccionado todos los elementos que se utilicen, se presiona *OK* y se tiene como se muestra en la figura 4.17

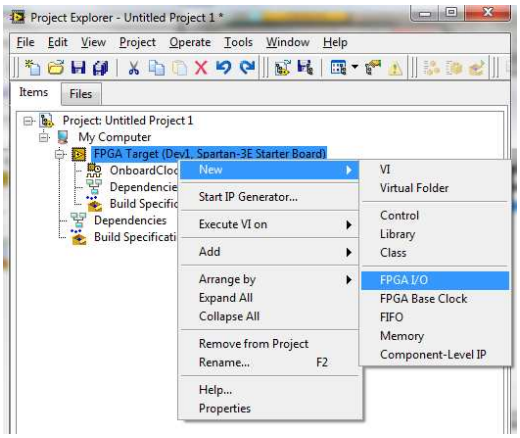


Figura 4.16 Agregar E/S

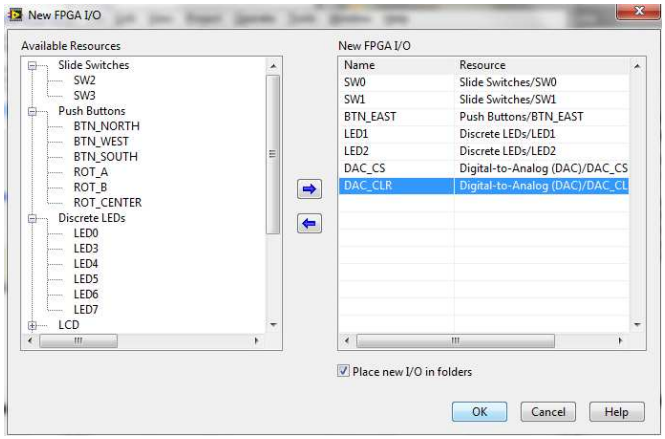


Figura 4.17 Selección de E/S

Estas entradas y salidas están agregadas al proyecto, ahora se las agrega al diagrama de bloques para su utilización:

1. Como ya se tiene el FPGA VI, se abre el diagrama de bloques, se arrastra y se suelta la E/S que se desee utilizar desde el *Project Explorer* hacia el diagrama de bloques. Figura 4.18

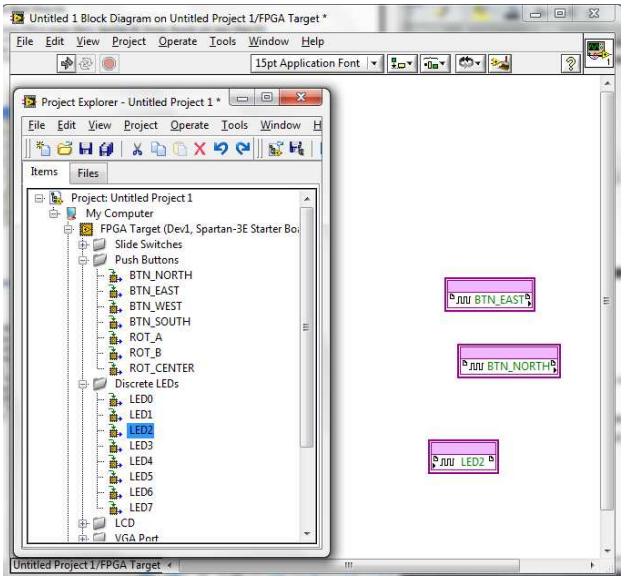


Figura 4.18 E/S agregadas al Diagrama de Bloques

Funciones de la Paleta FPGA

Cuando se selecciona un proyecto con FPGA, LABVIEW muestra solo las opciones disponibles para una FPGA específica, esto es, que solo muestra las funciones y subpaletas que se pueden utilizar con ese dispositivo.

Las *funciones* que incluye LABVIEW FPGA son: operaciones booleanas, comparaciones y operaciones matemáticas básicas.

4.5.2.1.3 Paleta de Funciones LabVIEW FPGA

Esta paleta ofrece todas las posibilidades de funciones que se pueden utilizar en el diagrama de bloques dentro del entorno de LabVIEW FPGA, donde al hacer click se escoje y ubica dentro del programa.

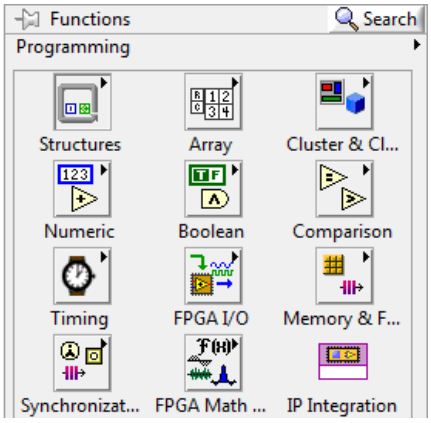


Figura 4.19 Paleta de Funciones [24]

Las Funciones contenidas en esta Paleta Functions>>Programming son:

- ***Structures,*** para el control del flujo de datos.
- ***Array,*** para crear y manipular conjunto de datos del mismo tipo y de tamaño fijo.
- ***Cluster & Class*** para crear y manipular conjunto de datos de diferente tipo y de tamaño fijo.
- ***Numeric,*** para realizar operaciones aritméticas de tipo entero con signo y sin signo.
- ***Boolean,*** para realizar operaciones lógicas.
- ***Comparison,*** para comparar valores booleanos, aritméticos, arrays y clusters.

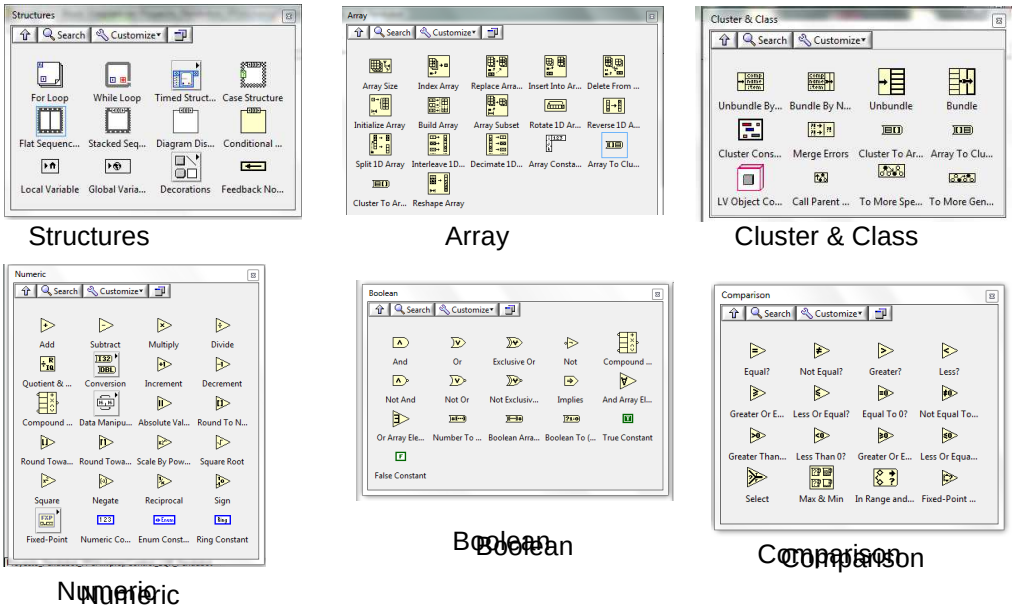


Figura 4.20 (a) Funciones de la paleta Functions

- ***Timing,*** para controlar el tiempo de ejecución de operaciones del FPGA.
- ***FPGA I/O,*** para realizar operaciones de lectura/escritura y configuraciones de los puertos de la tarjeta FPGA
- ***Memory & FIFO,*** para acceder a la memoria del FPGA y la función FIFO (first input-first output) para transferir datos.
- ***Synchronization,*** para sincronizar tareas de ejecución en paralelo y pasar datos entre tareas en paralelo.

- ***FPGA Math & Analysis,*** para realizar operaciones matemáticas de alto rendimiento (HighThroughput Math) y operaciones de control sobre el FPGA.

  Las operaciones de alto rendimiento permiten realizar operaciones de punto flotante.

  Las operaciones de Control permiten realizar análisis de señales, generar señales de onda cuadrada, senoidales y cosenoidales y crear aplicaciones de control (PID).
- ***Ip Integration Node,*** integra código IP (intellectual property), es decir se maneja código en VHDL dentro de LabVIEW como módulos adicionales. Antes de que se integre este código, es necesario que este previamente compilado.

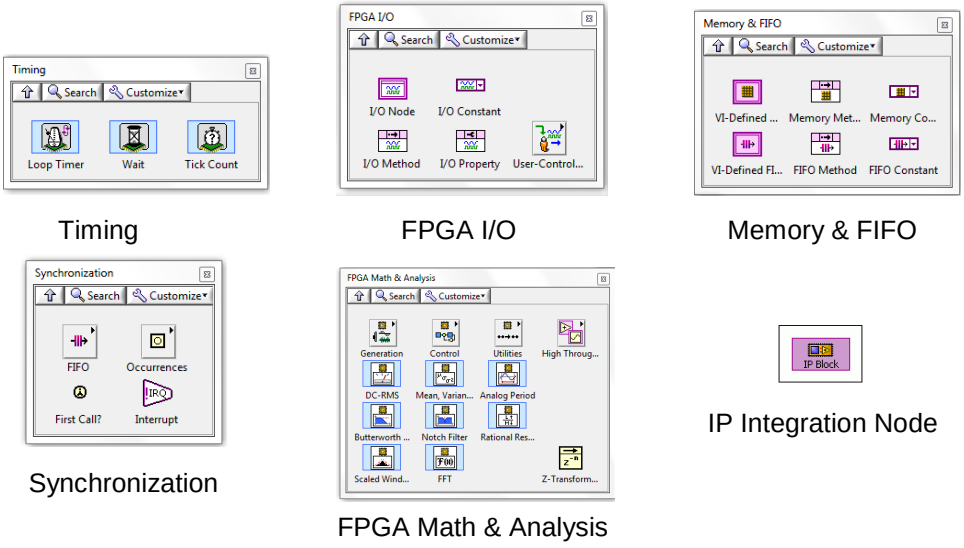


Figura 4.20 (b) Continuación de funciones de la paleta Functions

4.5.2.1.4 Creación HOST-FPGA

El Host se debe crear dentro del mismo proyecto del FPGA. Se siguen los siguientes pasos:

1. Se hace click-derecho sobre *My Computer* en la ventana del proyecto y se selecciona *New»VI.* Se nota que el nuevo VI se encuentra bajo el árbol de *My Computer.*

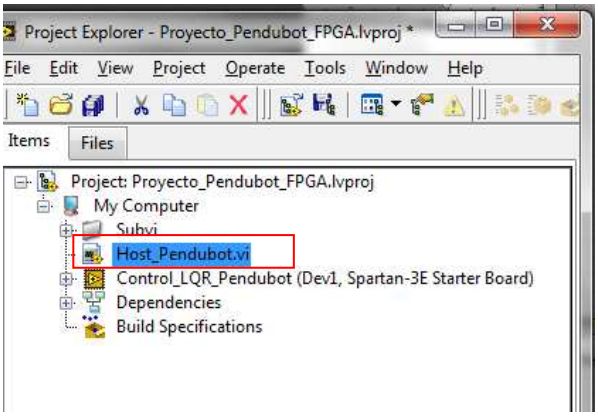


Figura 4.21 HOST_FPGA

2. Sobre el diagrama de bloques del nuevo VI, se coloca la función >>*Open FPGA VI Reference*, la función se encuentra en la paleta de funciones en la interfaz de FPGA. El *Open FPGA VI Reference* se utiliza para hacer referencia al FPGA-VI situado en el mismo proyecto. Sobre ésta función se hace click derecho y se selecciona el FPGA-VI, como se muestra en la Figura 4.23.

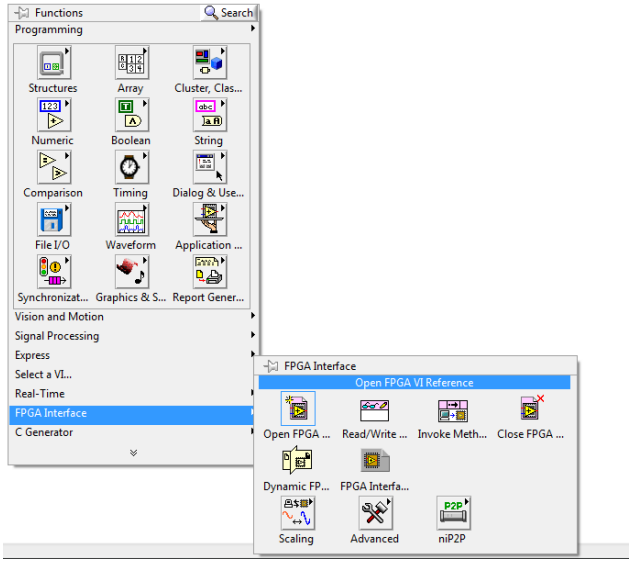


Figura 4.22 HOST_FPGA, *Open FPGA VI Reference*

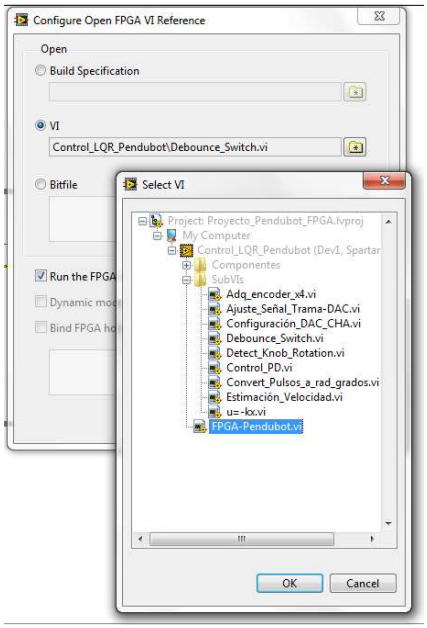


Figura 4.23 Selección del FPGA-VI

3. Se coloca la función *Function>>FPGA Interface>>Read/Write* el cual hace referencia a los terminales de control del FPGA-VI creado.
4. Se hace click sobre la función *Read/Write* y se selecciona los controles que se deseen observar y/o manipular en el Host VI.
5. Se cierra el recurso abierto en la función función *Function>>FPGA Interface>>Close FPGA VI Reference.*
6. El Host VI debe ser similar a la Figura 4.24.

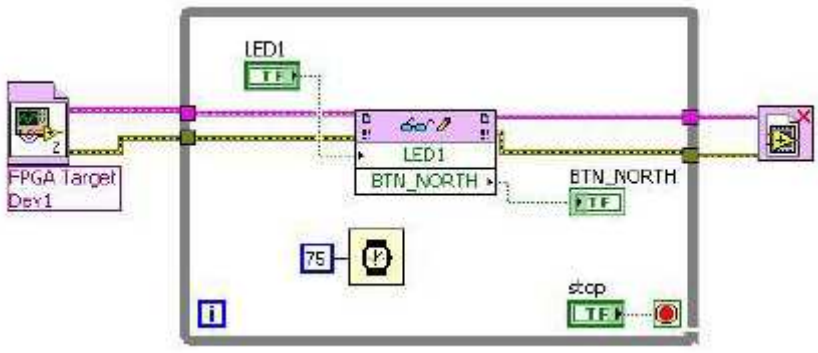


Figura 4.23 Host VI

Este VI permite manipular los controles del FPGA VI en la tarjeta de desarrollo.

Una vez que se programa la FPGA VI, el diseño se compilará y se descargará en la FPGA (Capítulo 6), y por lo tanto funcionará independiente de un computador. Dependiendo del diseño implementado, interactuará con el exterior de distinta manera y se podrá comprobar su correcto funcionamiento.

CAPÍTULO 5

IMPLEMENTACIÓN DEL ALGORITMO

En este capítulo se hace referencia a la programación en la tarjeta Spartan-3E de Xilinx y los aspectos a tomarse en cuenta para el desarrollo del programa, introduciéndose en cada una de las etapas que lo componen, esto es los algoritmos de adquisición de datos, pre-procesado, algoritmo de control, regulación del punto de operación y determinación de la salida del sistema, así como también la interfaz con el usuario y el módulo de comunicación con un computador para registrar los valores de las variables.

Adicionalmente se indica el hardware necesario para la interfaz física entre el Pendubot y el FPGA, ya que para llevar al Pendubot a las posiciones tope y media de equilibrio, se controla el par aplicado al motor, τ, y para el control del motor se utiliza un servoamplificador en donde la entrada de éste es adaptada para la Spartan-3E Starter Kit.

El programa de control y la interfaz de usuario son realizados en el Software LABVIEW v10.0. Al programar en LABVIEW FPGA, se diseñan diferentes *subVI's* que realizan una tarea específica dentro del FPGA, estos *subVI's* son interconectados en un *VI* general de mayor jerarquía que contiene toda la lógica necesaria para el funcionamiento del sistema.

5.1 ADQUISICIÓN DE SEÑALES

La posición de los eslabones del Pendubot se obtiene a través de encoders en cuadratura. Los encoders permiten obtener la posición, velocidad y el sentido de giro de los eslabones, por lo que se realizan diferentes Subvi's para obtener estas medidas físicas.

Los encoders son alimentados con 5V, pero el FPGA solo funciona con 3.3V ya que los pines de E/S de los puertos de expansión sólo permiten un máximo de

+3.3 V como 1L (estándar LVTTL), por lo que es necesario un circuito adicional que permita esta conversión de datos.

5.1.1 ACONDICIONAMIENTO DE SEÑALES DE ENCODER

Para la captura de las señales de los encoders se utilizará un circuito adicional para regular el voltaje de +5V a +3.3V que necesita la FPGA.

Para ello se adquirirán buffers para las entradas, ya que éste utiliza un solo voltaje de referencia (VDD) para todos los buffers, y las entradas en nivel alto pueden exceder el voltaje de referencia VDD cuando se lo utiliza como convertidor de niveles lógicos, es decir proporcionará una conversión de niveles lógicos para seguridad de la FPGA.

Los buffers que se adquirieron son los CD4050BC, que son circuitos integrados que contienen seis buffers no inversores internos, figura 5.1.

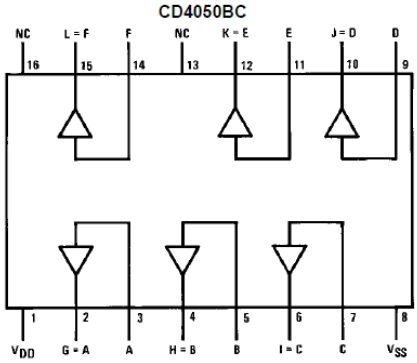


Figura 5.1 Configuración interna del CI CD4050BC [16]

Funcionamiento: El pin uno se lo alimenta con el voltaje de referencia de +3.3V del FPGA (conector J1 de la Tarjeta Spartan-3E) y cuando la entrada de un buffer reciba un voltaje lógico en alto de +5V, se tendrá a la salida como máximo un voltaje lógico en alto de +3.3V; permitiendo conectarla directamente como entrada a la Spartan-3E.

La figura 5.2 muestra parte de la interfaz física y el circuito utilizado que se diseña para el manejo de las entradas de los encoders al FPGA.

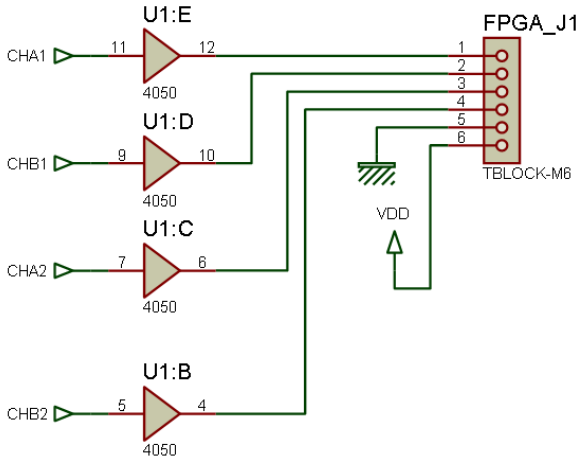


Figura 5.2 Circuito de manejo de los encoders.

5.1.2 PROGRAMA DE ADQUISICIÓN DE DATOS Y ESTIMACIÓN DE VELOCIDADES.

5.1.2.1 Subvi de adquisición de datos del encoder

Su función es capturar los pulsos desfasados de cada canal e interpretarlos para generar un conteo que indique al programa en qué posición se encuentra cada eslabón. Se presentará el diseño para un encoder incremental con resolución x4, es decir con resolución máxima. Este diseño tiene como propósito elevar la resolución de la señal de retroalimentación, empleando el mismo encoder, obteniéndose una resolución de 4096 pulsos por revolución. Se diseña totalmente con circuitos digitales para su implementación en lógica reconfigurable.

Para el caso x4 se debe tener en cuenta los flancos de ambas señales, figura 5.3. Para detectar los flancos se utiliza la compuerta XOR en la cual se conecta el estado actual y el anterior y con ello determinar el flanco de subida o bajada.

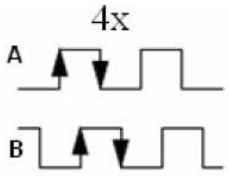


Figura 5.3 Flancos para 4x

Para determinar el sentido de giro se utiliza la señal actual del canal A y la señal anterior del canal B. El subvi implementado se indica en la figura 5.4. Para

entender mejor el funcionamiento se ha de pensar que cuando se cambia de dirección, lo que antes era el flanco de subida, pasa a ser flanco de bajada y viceversa.

- Si se tiene flanco de subida en A
  - Si B=0, se desplaza un pulso en sentido antihorario
  - Si B=1, se desplaza un paso en sentido horario

- Si se tiene flanco de bajada en A
  - Si B=1, se desplaza un paso en sentido antihorario
  - Si B=0, se desplaza un paso en sentido horario

- Si se tiene flanco de subida en B
  - Si A=1, se desplaza un paso en sentido antihorario
  - Si A=0, se desplaza un paso en sentido horario

- Si se tiene flanco de bajada en B
  - Si A=0, se desplaza un paso en sentido antihorario
  - Si A=1, se desplaza un paso en sentido horario

Para las velocidades angulares de cada eslabón, se realiza una estimación debido a que solo se tiene realimentación de las posiciones, para ello se ha empleado la derivada de la posición y para poder disminuir o eliminar el ruido numérico que provoca la derivada se ha utilizado un filtro pasabajos digital de Butterworth de segundo orden. La frecuencia de corte se ha elegido en forma experimental mediante pruebas realizadas en el funcionamiento del sistema. Se debe tomar en cuenta el periodo de muestreo.

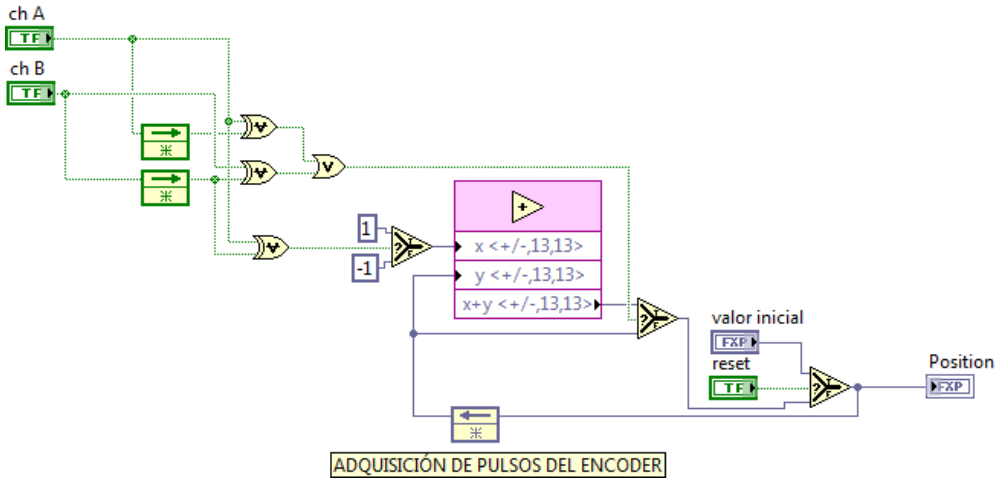


Figura 5.4 SubVI de adquisición de posición de los eslabones.

Icono del subvi 

Antes de realizar la estimación de las velocidades es necesario realizar la transformación de los pulsos a radianes de los eslabones.

5.1.2.2 Subvi de conversión de pulsos a radianes

Permite convertir de pulsos a radianes, se utiliza la siguiente relación:

$$\theta = \frac{\theta_{pulsos} * 2\pi}{4096} = \theta_{pulsos} * 0{,}001533 = \frac{\theta_{pulsos}}{651{,}8986}$$

Se utilizará de esta forma para reducir el número de multiplicaciones en las operaciones, se debe a que el número de estos operadores es limitado a 18 y al momento de compilar todo el programa da un error de programación.

La figura 5.5 muestra la implementación del subvi.

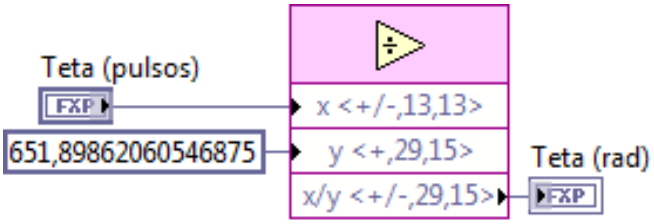


Figura 5.5 SubVI de conversión de pulsos a rad.

Icono del subvi 

5.1.2.3 Subvi de estimación de velocidad

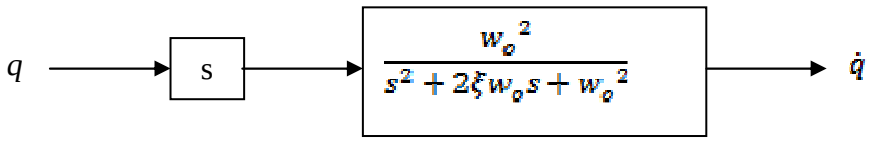
Como ya se menciono, para obtener las velocidades angulares se realizará una estimación a partir de las posiciones angulares.

El filtro que se utilizará está dado por la siguiente función de transferencia:

$$T(s) = \frac{{w_o}^2}{s^2 + 2\xi w_o s + {w_o}^2}$$

Donde: $w_o = 2\pi f \;; f = 20\ Hz, que\ es\ la\ frecuencia\ de\ corte$ y se utilizará un $\xi = \sqrt{2}/2$ para que el filtro tenga un comportamiento cercano al ideal.

El siguiente diagrama de bloques permitirá estimar la velocidad a partir de la posición angular



La función de transferencia obtenida es:

$$\frac{\dot{q}}{q} = \frac{{w_o}^2}{s^2 + 2\xi w_o s + {w_o}^2}$$

Discretizando la función de transferencia utilizando el método de *tustin*, se obtiene la expresión que permitirá estimar la velocidad angular.

$$\dot{q_k} = 22.26337(q_k - q_{k-2}) + 1.32079\ \dot{q_{k-1}} - 0.49889\ \dot{q_{k-2}}$$

Se implementará esta ecuación en un subvi como se indica en la figura 5.6

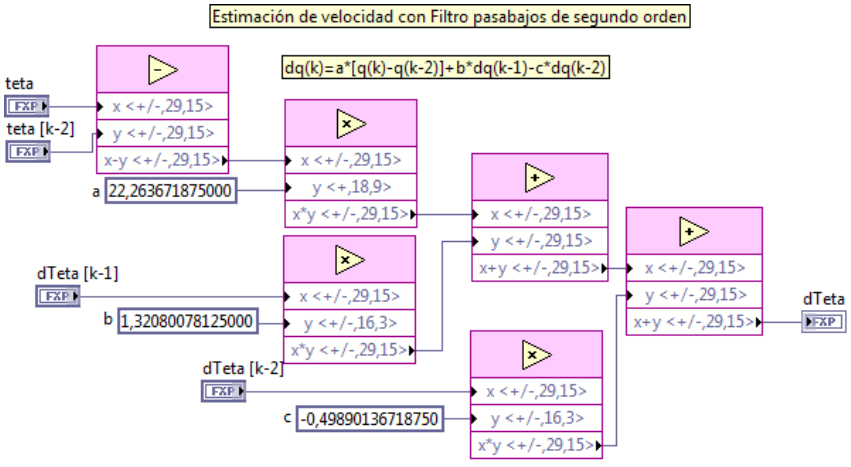


Figura 5.6 SubVI de estimación de velocidad angular con filtro pasabajos de segundo orden, icono del subvi 

Una vez obtenidas las señales de las variables de estado, se continúa con la programación de los controladores del Pendubot, los cuales permitirán mantener en equilibrio al sistema.

5.2 CONTROL DEL PENDUBOT

El control del Pendubot en las posiciones tope y media de equilibrio, se realiza empleando la técnica LQR (Regulador cuadrático lineal). Para ello se utilizará un VI en LABVIEW que permitirá determinar el vector de control óptimo ***K*** de este sistema en cualquier posición.

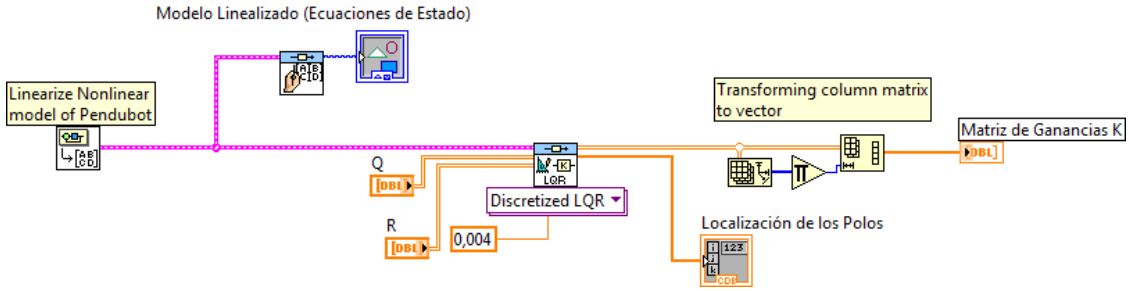


Figura 5.7 SubVI para determinar vector ***K***

Este VI está implementado en el Host de Interface con la PC y no se descargará al FPGA.

Los valores de la matriz Q se manipulan para ponderar la importancia relativa de los componentes del vector de estado, y corresponden a las posiciones de los eslabones, esto es los estados x1 y x2. Los valores de la matriz R se manipulan para ponderar el gasto de energía en la acción de control, esta matriz tiene una relación inversa con la magnitud de la señal de control, es decir, mientras mayor sea la ponderación de la matriz menor será la señal de control.

La señal de control que se requiere es:

$$u(k) = -K * x(k)$$

5.2.1 SUBVI DE CONTROL LQR

El controlador LQR es el encargado de hacer que cada uno de los eslabones se mantengan en las posiciones de equilibrio dadas en la linealización del sistema, para ello se ha implementado el lazo de control mostrado en la Figura 5.8, en el

cual se realimentan la posición y velocidad de cada eslabón mediante los subvi's de adquisición de datos y estimación de velocidades dadas en la sección anterior.

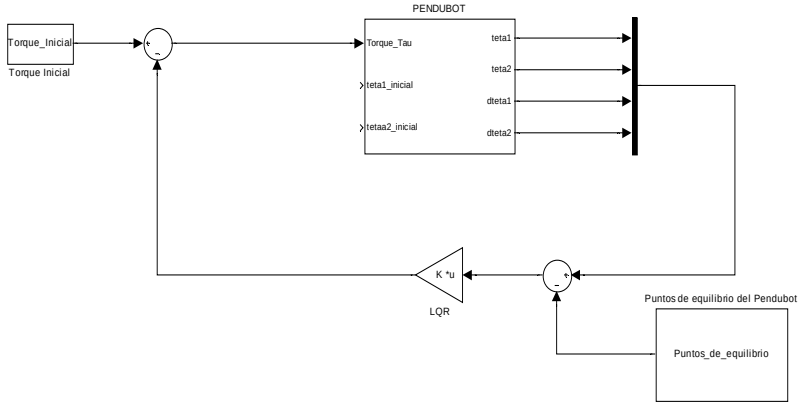


Figura 5.8 Esquema de control del sistema

Como la señal de control es $u(k) = -K * x(k)$, entonces se realizará el subvi que realiza esta operación, se indica en la Figura 5.9

Se implementará utilizando un lazo *for*, de esta forma también permitirá reducir los operadores dentro del algoritmo de control. Este subvi permite mantener estable al Pendubot en la posición de equilibrio inestable. Para poder llevar los eslabones a esta posición se utiliza el control PD el cual consiste en llevar los eslabones desde su posición de equilibrio estable a las posiciones inestables, a través de los movimientos del primer eslabón.

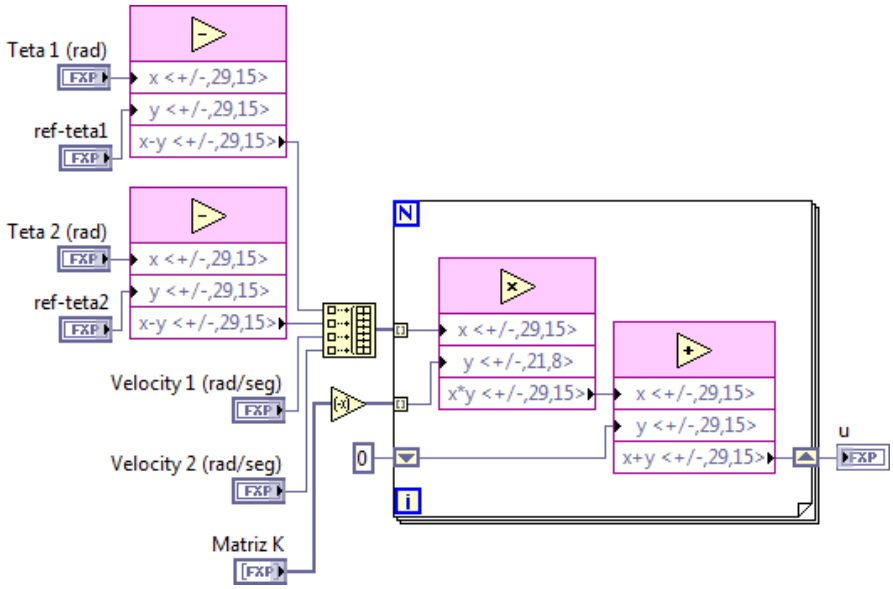


Figura 5.9 Subvi del control LQR

Icono del subvi 

5.2.2 SUBVI CONTROL PD

Este subvi realizará la operación del control PD para el balanceo del Pendubot, es decir para llevar los eslabones a los puntos de equilibrio que se desee controlar.

En base a la ecuación 3.8 que se escribe nuevamente, se realiza la programación del controlador PD con realimentación parcial, como sigue, figura 5.10.

$$v_1 = \ddot{\theta}_1^d + k_d(\dot{\theta}_1^d - \dot{\theta}_1) + k_p(\theta_1^d - \theta_1)$$

Cada uno de estos controladores emite un valor numérico el cual debe ser la señal del actuador (motor) para poder mantener estable al sistema, y para tener la señal física se utiliza el conversor digital análogo del FPGA.

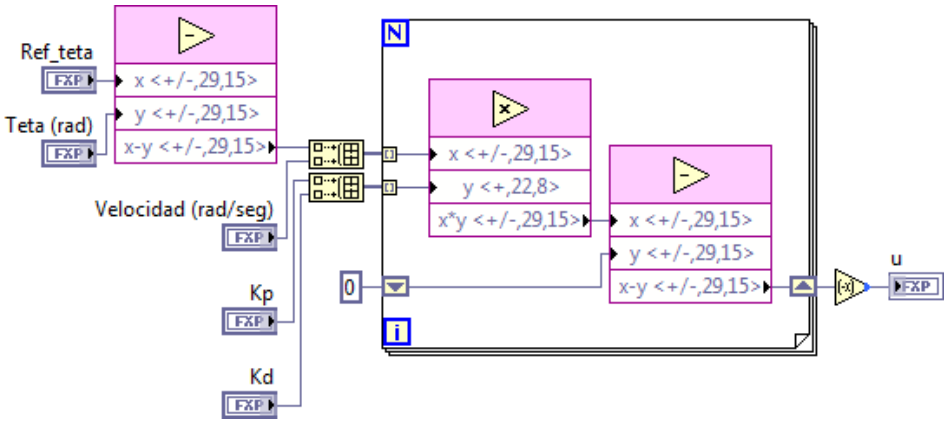


Figura 5.10 Subvi del control PD

Icono del subvi 

5.3 SALIDA DE SEÑALES

La tarjeta Spartan-3E Starter Kit incluye un conversor Digital-Análogo de cuatro canales, con comunicación SPI (Serial Peripheral Interface), de 12 bits de resolución con una salida de 0 a 3.3 V o de 0 a 2.5 V, dependiendo de la salida configurada.

5.3.1 SUBVI DEL CONVERSOR DIGITAL-ANÁLOGO (DAC)

Se utiliza para convertir la señal de control digital a una señal de control analógica.

Para utilizar el DAC, se realizarán dos Subvi's, el primero para la Configuración del DAC que solo actuará al encender el FPGA, es decir funciona únicamente cuando empieza el programa y el segundo Subvi que actúa permanentemente, este ajusta la señal de control y la envía al DAC por comunicación SPI.

5.3.1.1 Subvi de Configuración

El DAC de la tarjeta Spartan-3E Starter Kit de Xilinx se comunica a través del bus SPI. Como el bus SPI es compartido con otros dispositivos del módulo, es importante que estos se deshabiliten cuando se utilice la comunicación entre el FPGA y el DAC de acuerdo con la tabla 4.2; y, en la tabla 4.1 se muestra la lista de señales que se deben configurar (se indican las tablas nuevamente en Tabla 5.2 y 5.1 respectivamente).

Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA→DAC	Serial data: Master Output, Slave Input
DAC_CS	N8	FPGA→DAC	Active-Low chip-select. Digital-to-analog conversion starts when signal returns High.
SPI_SCK	U16	FPGA→DAC	Clock
DAC_CLR	P8	FPGA→DAC	Asynchronous, active-Low reset input
SPI_MISO	N10	FPGA←DAC	Serial data: Master Input, Slave Output

Tabla 5.1 Señales de Configuración [9]

Signal	Disabled Device	Disable Value
SPI_SS_B	SPI serial Flash	1
AMP_CS	Programmable pre-amplifier	1
AD_CONV	Analog-to-Digital Converter (ADC)	0
SF_CE0	StrataFlash Parallel Flash PROM	1
FPGA_INIT_B	Platform Flash PROM	0

Tabla 5.2 Lógica para deshabilitar los dispositivos compartidos con el bus SPI [9]

El subvi de configuración del DAC se lo realizará de acuerdo a las tablas anteriores y se muestra en la figura 5.11.

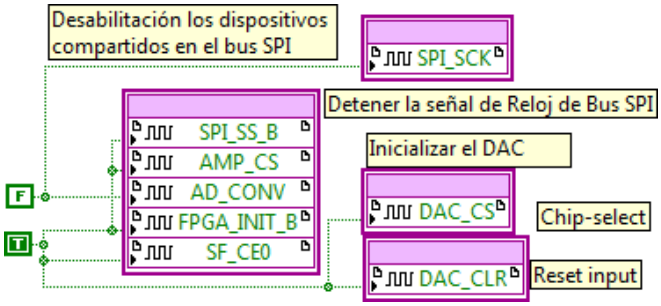


Figura 5.11 SubVI de configuración del DAC.

Icono del subvi 

5.3.1.2 Subvi de Ajuste de señal y Trama DAC

El subvi ajusta la señal de control y realiza la trama SPI para la comunicación entre el DAC y el FPGA de la tarjeta Spartan-3E.

La figura 5.12 muestra un ejemplo de los tiempos del bus SPI. Cada bit es transmitido o recibido con la señal de reloj SPI_SCK, este bus soporta el reloj del FPGA que es de 50 MHz, sin embargo se puede explorar los tiempos de comunicación en el data sheet del conversor LTC2624 [17].

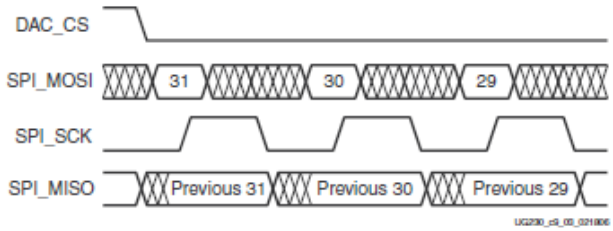


Figura 5.12 Formas de onda de la comunicación SPI [9]

El protocolo de comunicación requerido para la interfaz entre el DAC y el FPGA se muestra en la figura 5.13. Como se observa el protocolo es de 32-bits y los datos son transmitidos respetando los tiempos de comunicación. La transmisión es bit a bit y en flanco positivo del reloj, enviando primero el bit más significativo (MSB) y termina en el bit menos significativo (LSB).

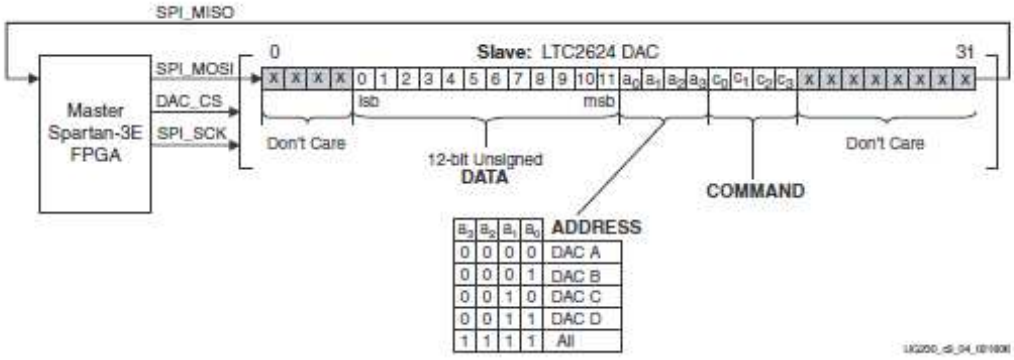


Figura 5.13 Protocolo de comunicación para el DAC LTC2624 [9]

Este subvi tendrá un limitador de la señal de control, debido a que solo se realizará el control entre los valores de -3 a +3 V, respetando las recomendaciones dadas en [3], el cual indica los límites de la señal de control con la finalidad de minimizar el esfuerzo producido por el actuador.

El Ajuste de la señal de control se realiza debido a que el DAC de la tarjeta Spartan-3E no emite valores negativos, por lo que es necesario realizar un ajuste de la señal, transformando a valores positivos y en proporción a la unidad, para que en la salida se tenga una relación lineal de los valores respecto al valor máximo entregado por el DAC, esto es:

$$y = (x/6 + 1/2)*4095$$

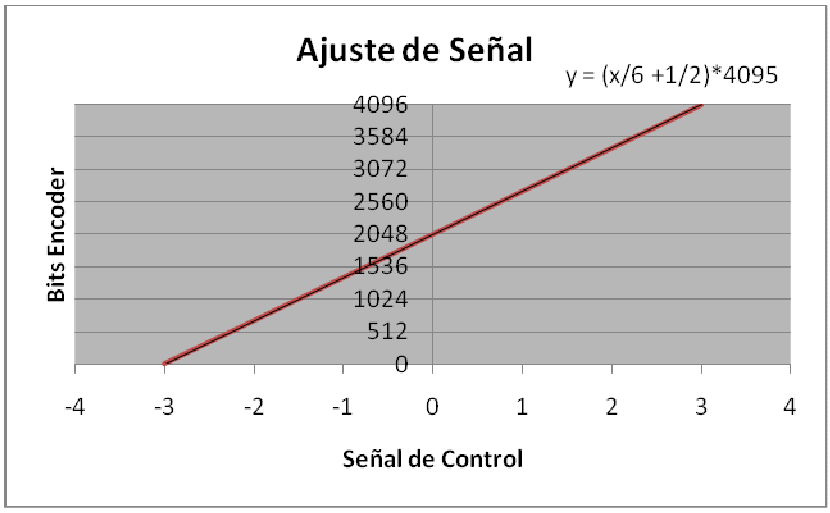


Figura 5.14 Ajuste de Señal de Control

En la figura 5.15 se muestra el subvi que realiza el ajuste de la señal de control y su relación a los 12 bits del DAC, además de la trama para la comunicación.

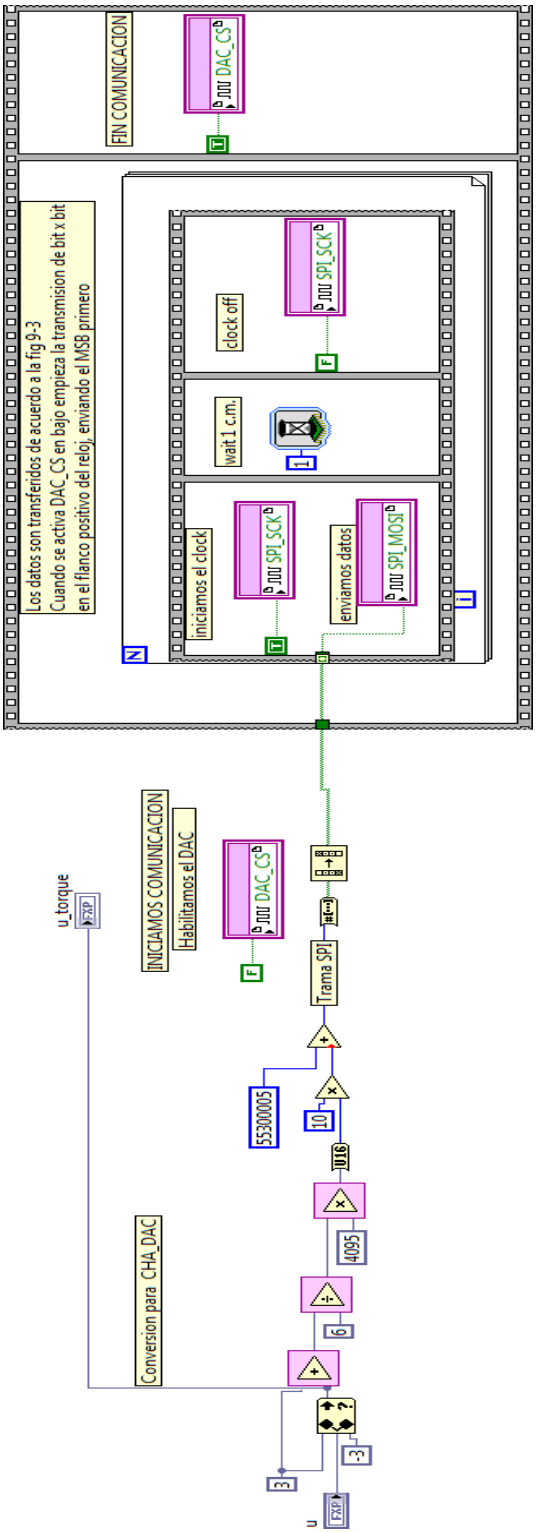


Figura 5.15 SubVI de ajuste y trama del DAC

Iicono del subvi 

El DAC entrega una señal positiva de 0 a 3.3V, que tiene una relación lineal con la señal de control y para tener nuevamente la señal de control original se realizará la transformación de voltaje, es decir de -3 a +3V, de acuerdo a la señal dada por el algoritmo de control.

5.3.2 ACONDICIONAMIENTO DE LA SEÑAL DE CONTROL

Para la señal de control se diseñó un circuito adicional que permite convertir la señal de salida del conversor digital análogo de 0 a +3.3 V en una señal adecuada de -3V a 3V para el control del motor dc del Pendubot, el cual se conectará a la entrada del sevoamplificador.

El circuito consta de tres etapas: la primer etapa es un amplificador seguidor, el cual permite acoplar el FPGA con el circuito de acondicionamiento, la segunda etapa consta de un amplificador no inversor y la tercer etapa de un sumador no inversor que permite restar un voltaje para obtener el valor deseado de voltaje, de acuerdo a la siguiente relación:

$$y = 1.8182\,x - 3 \qquad (5.1)$$

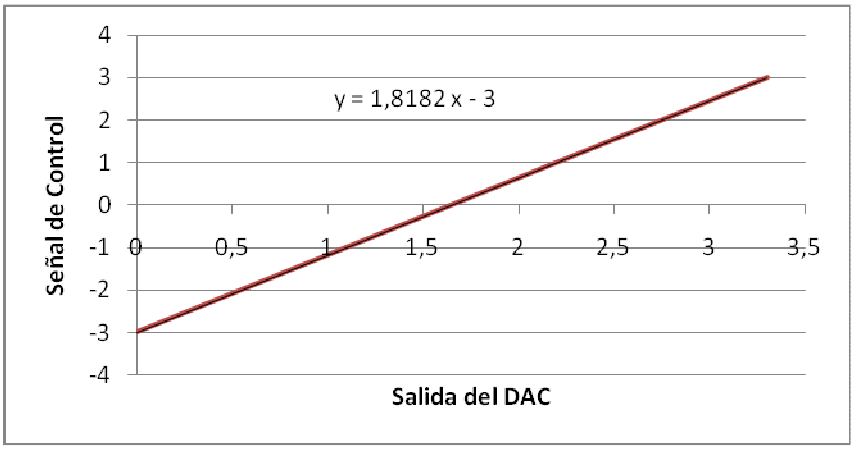


Figura 5.16 Señal de control

A continuación se da una descripción del diseño de las tres etapas mencionadas:

- La etapa de entrada para el voltaje de control, está conformada por un amplificador seguidor.

El objetivo de esta etapa es presentar una elevada impedancia de entrada a la señal de control. Esta etapa permite además que la señal de control no sea referenciada al circuito de acondicionamiento del servoamplificador con la finalidad de proteger a la tarjeta Spartan-3E Starter Kit en caso de falla.

- La segunda etapa consiste en un amplificador no inversor, cuya ganancia está ajustada de acuerdo a la ecuación 5.1 que es de 1.8182, la ganancia de voltaje está dada por:

$$G_2 = 1 + \frac{R_1}{R_2}$$

Donde $R_1 = 180\ \Omega$, y

$R_2 = 220\ \Omega$

$R_3 = R_1 \parallel R_2 = 99\ \Omega \approx 100\ \Omega$

- La tercera etapa consiste en un amplificador sumador no inversor, el cual permite restar un valor de voltaje para obtener el valor deseado para el control del sistema, de acuerdo a la ecuación 5.1 que es de -3 V. La ganancia de esta etapa es igual a 1.

$$V_O = V_1 \frac{R_6}{R_4} + V_2 \frac{R_6}{R_5}$$

Donde $R_4 = R_5 = R_6 = 3.3\ K\Omega$, para obtener:

$$V_0 = V_1 + V_2$$

Donde V_1 es la salida de la etapa anterior y V_2 es -3V.

La señal de -3V se tomó de una fuente de voltaje de -5V, realizada por un regulador de voltaje 7905 de acuerdo a las especificaciones del fabricante (datasheet 7905). Ya que el voltaje de salida en condiciones normales es

de -5V, es necesario implementar un divisor de tensión mediante el potenciómetro RV1 de tal forma que se obtenga el valor deseado de -3V.

El circuito diseñado se muestra en la figura 5.17

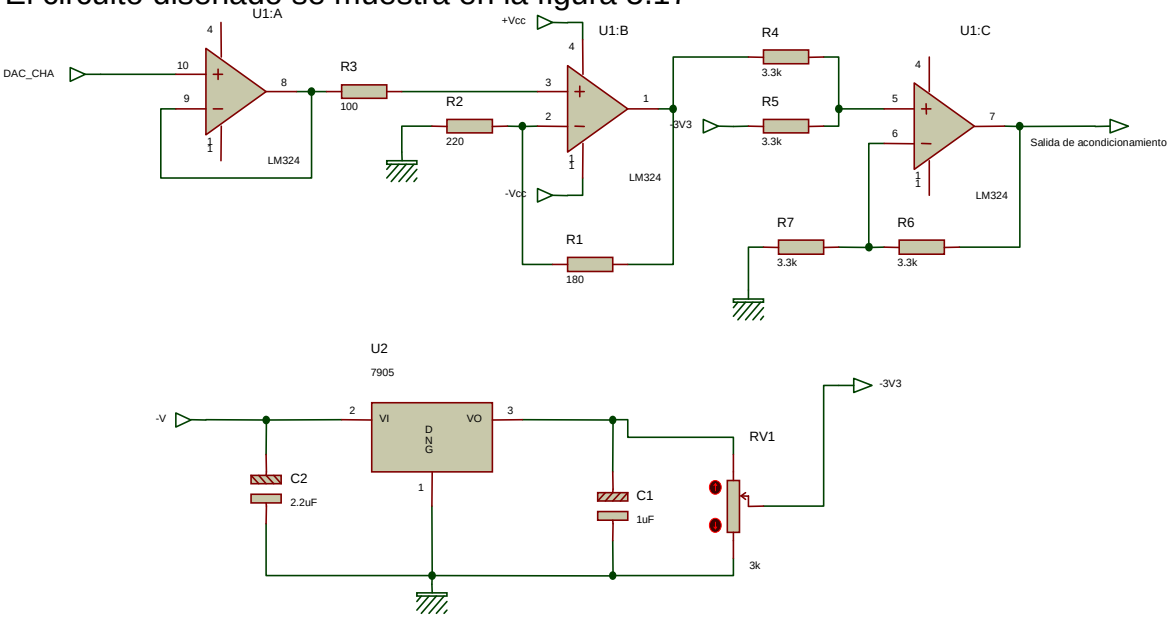
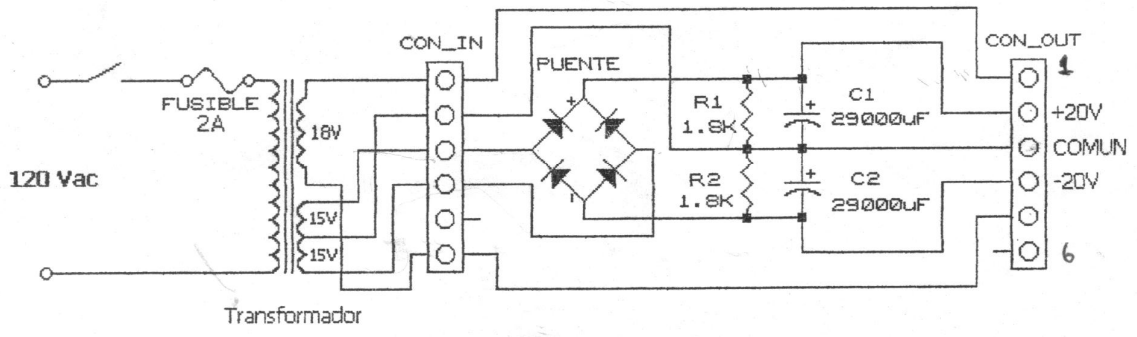


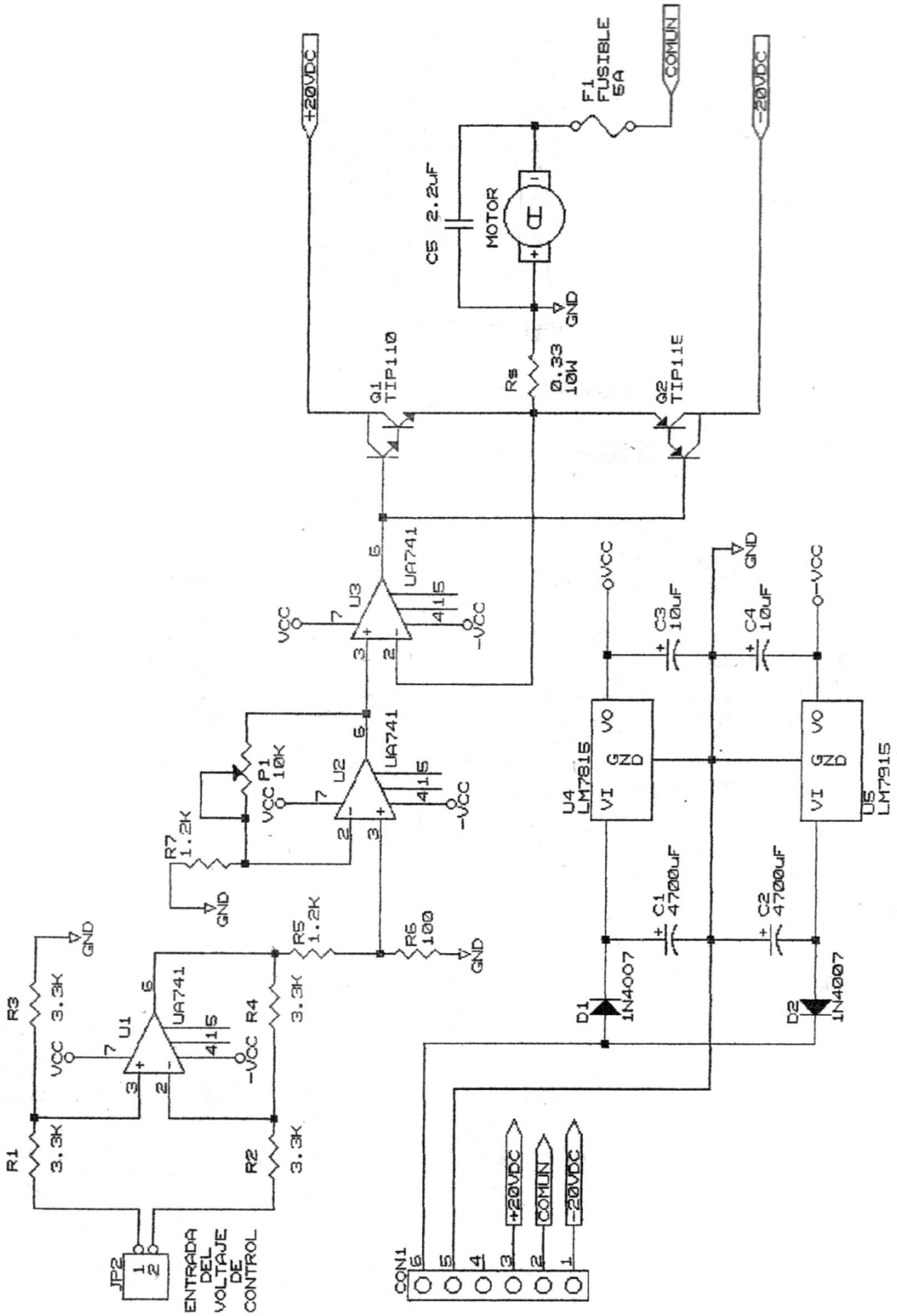
Figura 5.17 Circuito de acondicionamiento de voltaje

La fuente de ± 15V utilizada para la alimentación de los amplificadores operacionales que realizan el acondicionamiento de la señal de control, se la realiza con la del circuito de alimentación del servoamplificador (módulo de potencia del Pendubot) [3], el cual cuenta con éstas fuentes.

El circuito de la Fuente de poder y del servoamplificador se muestran en la figura 5.18 (a) y (b) respectivamente, a las cuales no se les realizarán modificaciones de los circuitos originales.



(a)



(b)

Figura 5.18 (a) Fuente de Poder [3]

(b) Circuito del Servoamplificador [3]

Con todas las señales disponibles se realizará la lógica del programa, utilizando el potencial del FPGA se realizará el programa en forma modular, para que cada uno de los módulos realice una función específica.

5.4 LÓGICA DEL PROGRAMA

El procesamiento de la FPGA es en paralelo, por lo que cada uno de los módulos puede funcionar independiente y a diferentes tiempos de ejecución.

5.4.1 MÓDULO DE ADQUISICIÓN

La adquisición de datos de realizará por medio del conector J1 y su ejecución dentro de un lazo *Timed Loop*, el cual permite la ejecución dentro de un ciclo de máquina es decir cada 20ns (f=50MHz), permitiendo adquirir los pulsos de forma rápida y precisa.

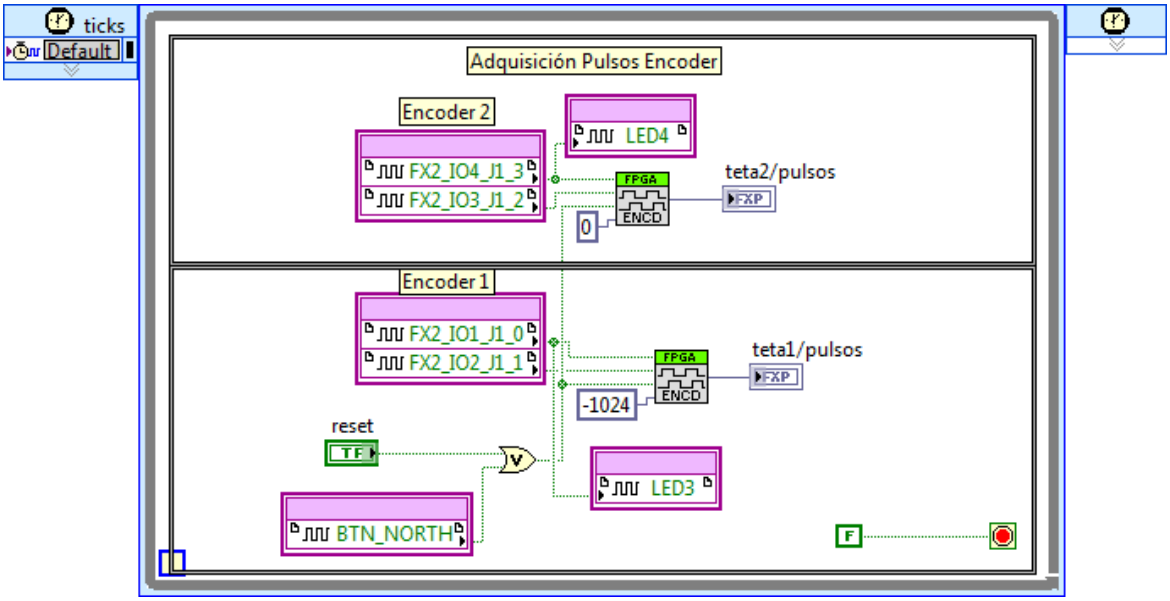


Figura 5.19 Modulo de Adquisición de Datos

5.4.2 MÓDULO DE CONVERSIÓN A RADIANES Y ESTIMACIÓN DE VELOCIDAD.

La conversión a radianes y la estimación se la realizará en un mismo lazo, para ello se utilizará un lazo for loop, reduciendo así la memoria utilizada del FPGA.

Las operaciones algebraicas se realizarán dentro de un lazo While Loop, debido a que las operaciones toman más de un ciclo de máquina en realizarse.
La ejecución de este lazo se realizará cada 200000 ticks (ciclos de máquina) lo que equivale a 4 ms que corresponde al tiempo de muestreo del sistema.

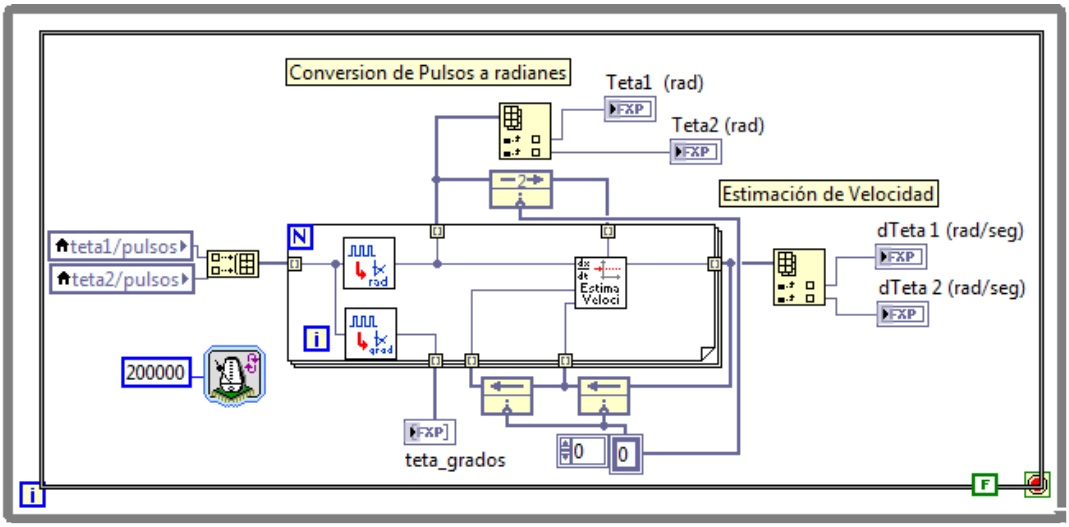


Figura 5.20 Modulo de Conversión de ángulos
y estimación de velocidad.

5.4.3 MÓDULO DE CONTROL LQR Y PD

Este módulo permitirá calcular el valor de la señal de control necesaria para el funcionamiento adecuado del sistema.
La forma de programación se realizará teniendo en cuenta el número de operaciones, el tamaño de la lógica y el mejor funcionamiento del sistema. Para ello se realizará la lógica dentro de un lazo *Timed Loop,* lo que permite reducir de forma substancial el tamaño de memoria del FPGA y las operaciones algebraicas se realizarán en conjunto dentro de un lazo *While Loop* por las razones explicadas anteriormente.

La ejecución de este lazo se realizará cada 200000 ticks lo que equivale a 4 ms al igual que el módulo de transformación de ángulos y estimación de velocidad.

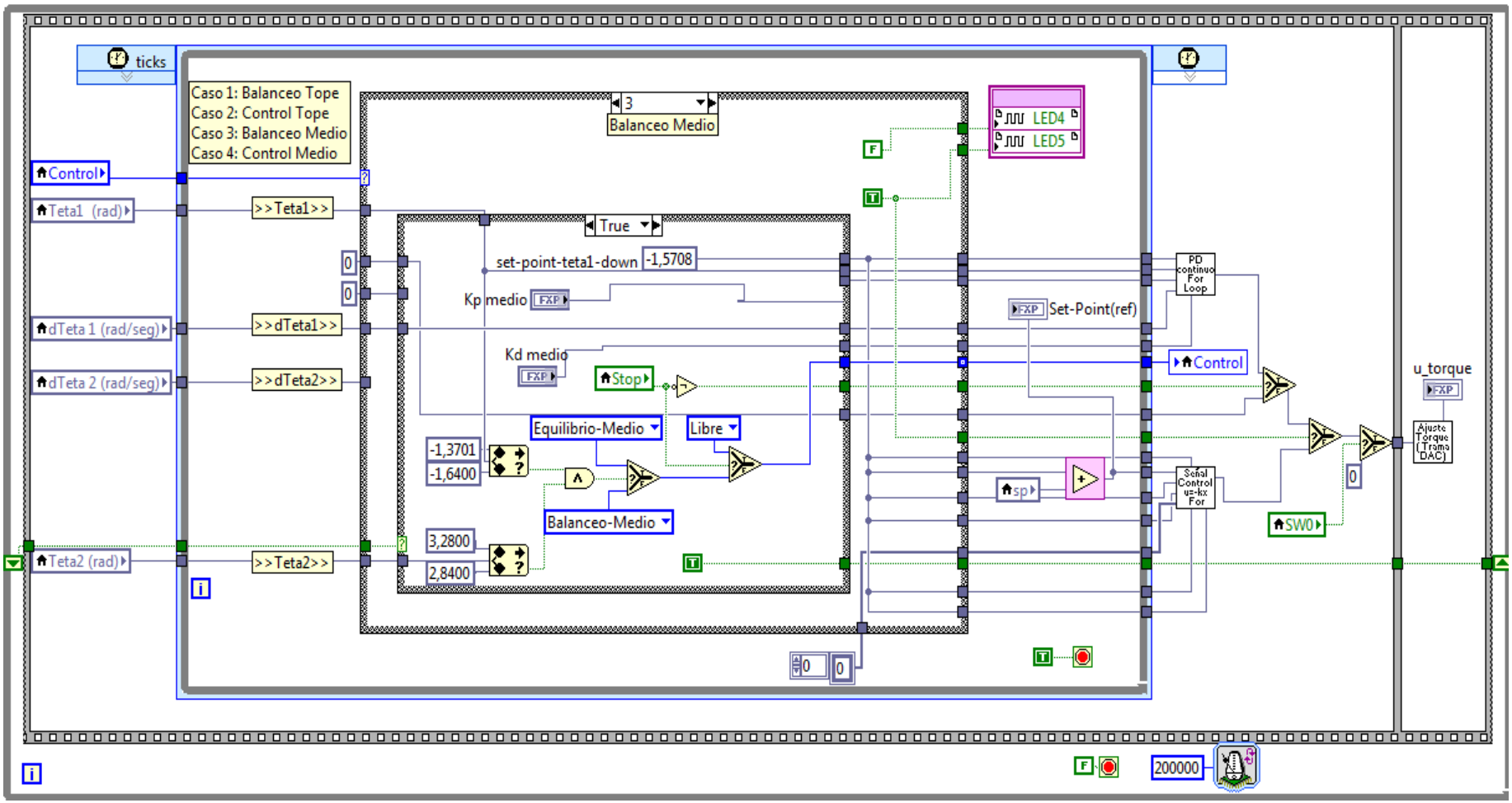


Figura 5.21 Modulo de control PD y LQR

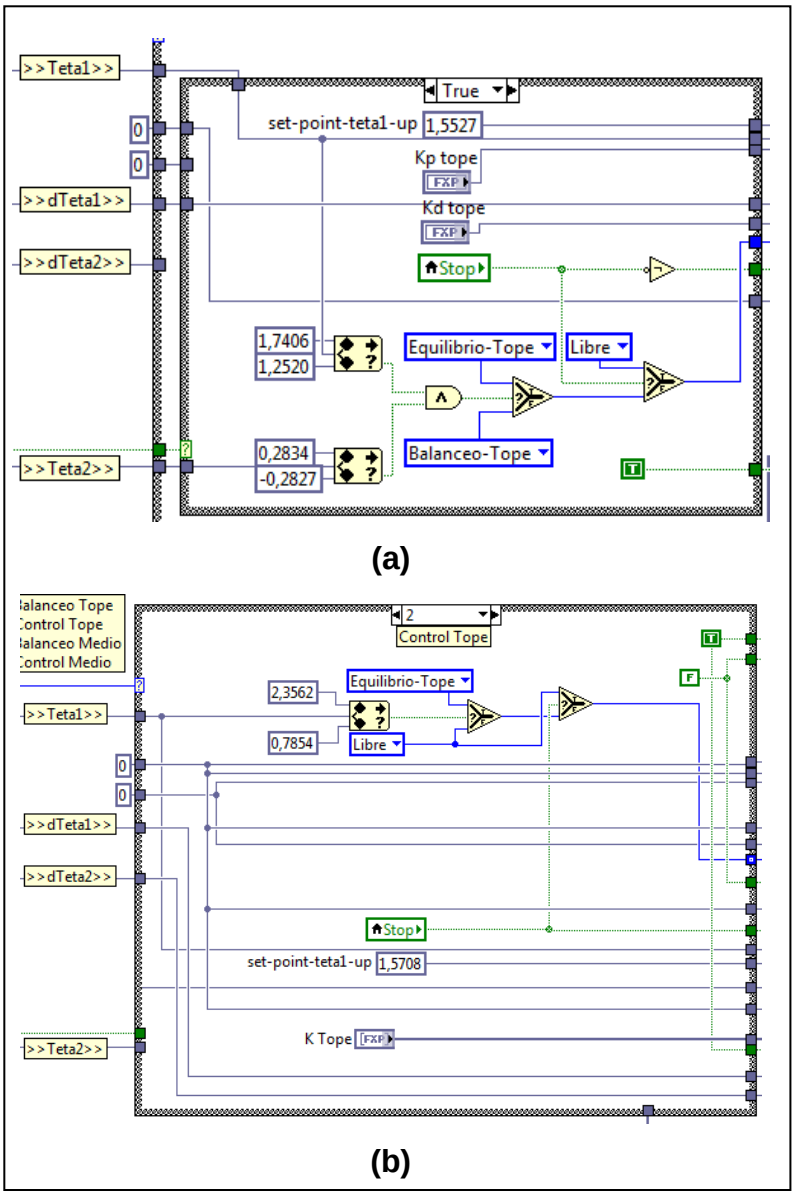
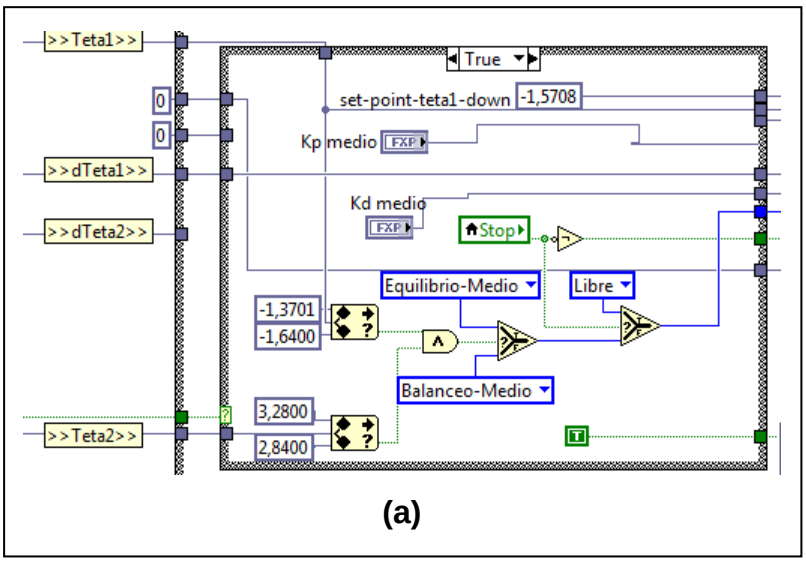


Figura 5.22 Control Tope

(a) Control Balanceo (b) Control Equilibrio



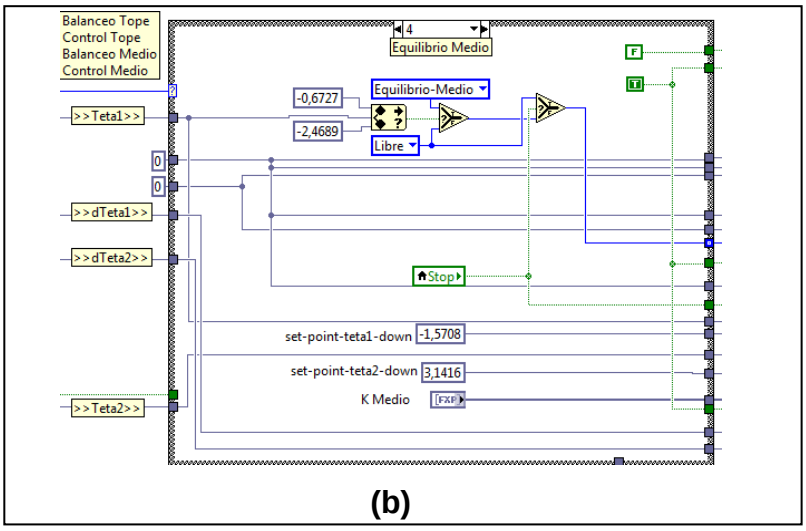


Figura 5.23 Control Medio

(a) Control Balanceo (b) Control Equilibrio

5.5 INTERFAZ DE USUARIO.

La programación de interfaz de las posiciones tope y media, se realizará mediante el programa computacional *LabVIEW* v.10.0. Además, se diseñó una interfaz desde la PC que cuenta con controles e indicadores para la ejecución del control tope y control medio y una interfaz desde el FPGA en el que se puede realizar ambos controles y variar el set-point.

Se ha buscado crear un diseño que sea fácil de utilizar y manipular.

5.5.1 INTERFAZ DESDE PC (HOST_PENDUBOT).

El programa de control desde PC permite al usuario manejar el Pendubot de manera sencilla, además muestra las posiciones de cada eslabón en gráficas respecto a la señal del set-point e indica la señal de control también gráficamente.

Este programa constará de varias etapas que en conjunto permiten un mejor manejo del sistema, con lo cual permite variar el punto de equilibrio a los alrededores del punto linealizado.

La pantalla principal se muestra en la figura 5.24 y la pantalla de control se muestra en la figura 5.25

La interfaz de usuario del computador tiene los siguientes controles:

1. Manual/Automático.
2. Control Medio.
3. Control Tope.
4. Detener el Control
5. Continuar/Regresar Pantalla.
6. Variación del Punto de Operación (Set-point)
7. Modelo Linealizado
8. Control PD
9. Encerar o reset Encoders.
10. Reset matriz de ganancia *K*
11. Led Indicador de sistema en Línea
12. Stop Interfaz.
13. Manual de usuario

Figura 5.24 Pantalla principal

El control del Pendubot en las posiciones tope y media de equilibrio, como se indica en el apartado 5.2, se realizará empleando la técnica LQR (Regulador cuadrático lineal). Para ello se utilizará un VI en LABVIEW que permitirá determinar el vector de control óptimo ***K*** de este sistema en cualquier posición. El cual está integrado en el HOST principal de control como se indica en la figura 5.25, al presionar el Botón de Modelo Linealizado.

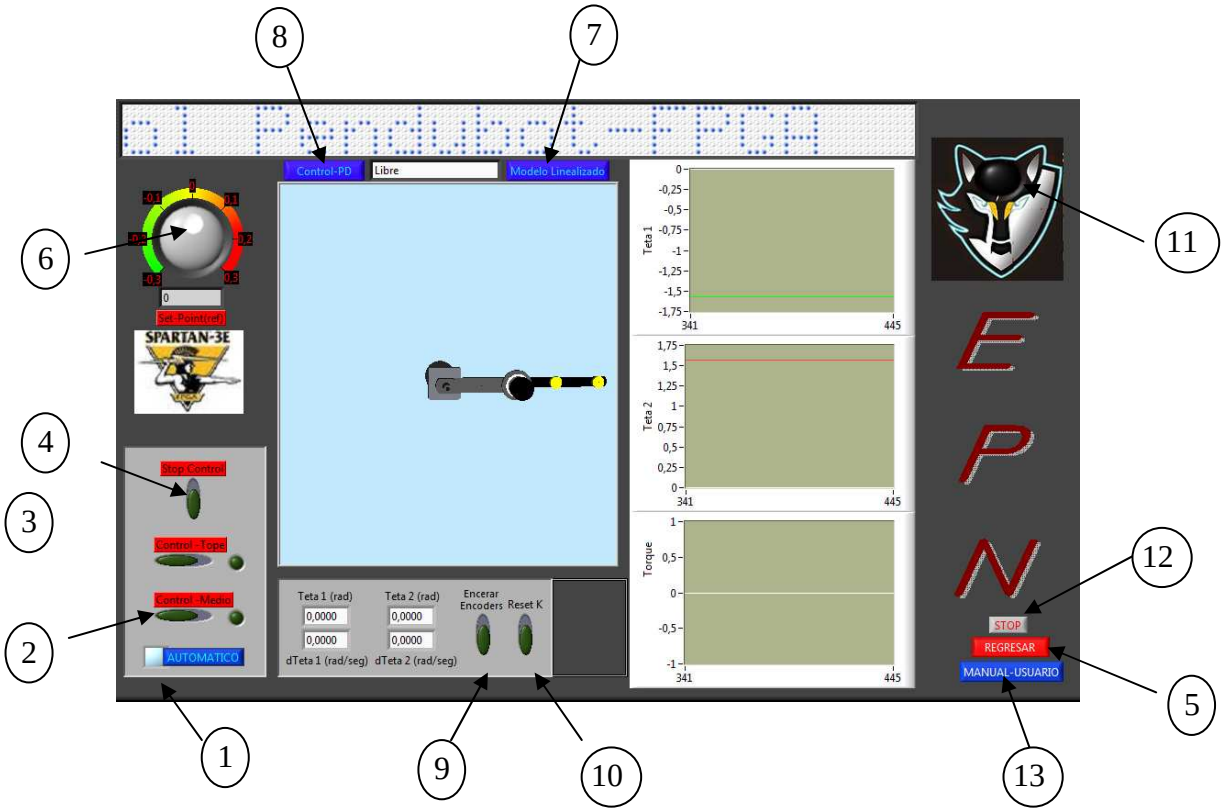


Figura 5.25 Host_FPGA y controles de interfaz

El botón Modelo Linealizado, figura 5.26 (a), permite observar las Ecuaciones de Estado del Modelo Linealizado en un punto de operación, además permite variar las matrices de ponderación y así modificar la matriz de Ganancias *K* del control LQR del sistema, asimismo se puede cambiar el punto de linealización del sistema y poder así tener diferentes puntos de operación, figura 5.26 (b).

El botón Control PD permite observar el controlador PD utilizado para el balanceo del sistema, además de poder ajustar las constantes *Kp* y *Kd*, figura 5.27

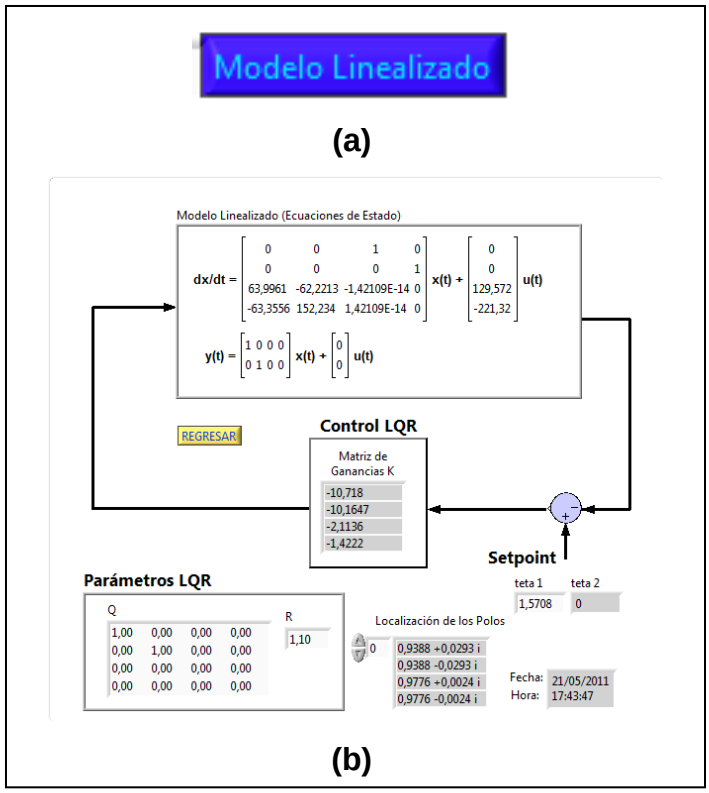


Figura 5.26 (a) Botón de Modelo Linealizado
(b) Panel Frontal para determinar vector ***K***

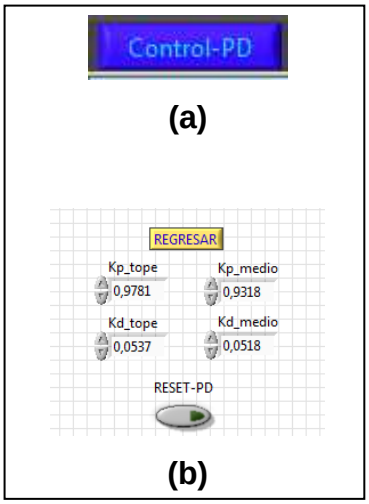


Figura 5.27 (a) Botón de Control-PD
(b) Panel Frontal de Control PD

Es importante que el dispositivo que maneja directamente el sistema, desde adquisición la datos, procesado y control sea también utilizado como interfaz de

usuario, sin necesidad de estar conectado a un computador, por ello se realizará una interfaz con los recursos de la tarjera Spartan-3E Starter Kit.

5.5.2 INTERFAZ DESDE TARJETA SPARTAN -3E STARTER KIT

Para realizar la interfaz desde este dispositivo, es necesario realizar subvi's adicionales que permitan configurar y manejar los recursos dados por esta tarjeta.

5.5.2.1 Subvi de Control de la Perilla

La Perilla permitirá variar el set-point, se ha utilizado un VI de detección de sentido de giro de la perilla, así como también un VI de antirrebote, (Detect_Knob_Rotation.VI [19] y Debounce_Switch.VI [18]), el cual se ha adaptado para realizar la función descrita.

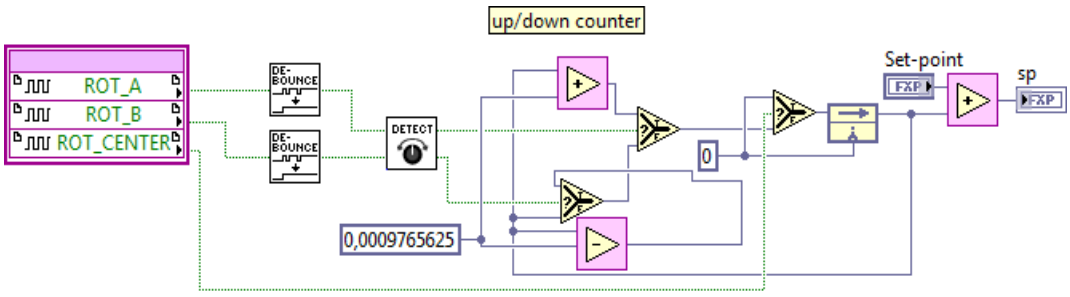


Figura 5.23 Control del Encoder Rotatorio

- *Debounce*.- Es un subvi de antirrebote.
- *Detect*.- Subvi que realiza la detección del sentido de giro de la perilla.

5.5.2.2 Manejo del LCD

Para el manejo del LCD se realizará la programación en VHDL, ya que LABVIEW FPGA permite la integración entre IP (Intellectual Property) y programación gráfica.

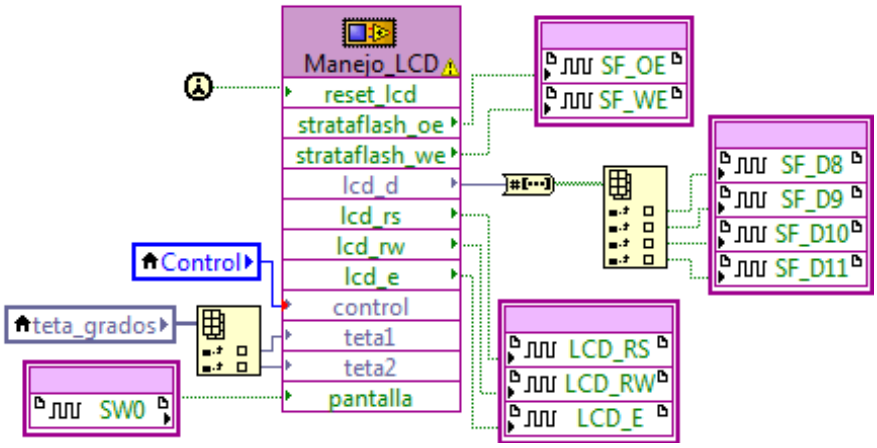


Figura 5.24 Manejo del LCD

Para realizar esta integración es necesario tener el código en vhdl, y se realizará en el software de Xilinx ISE Design Suite 12.1 (para mayor información de programación en vhdl referirse a [20] y [21]). El potencial de este dispositivo permite instanciar[1] microprocesadores para programación en Assembler, en el tutorial de "Integrating a picoblaze processor in LabVIEW FPGA by use of IP node" [22], indica cómo realizar este proceso y los programas utilizados para ello. El programa en Assembler del LCD y el programa en vhdl se indican en el Anexo D

5.5.2.3 Manejo de la Interfaz en la Tarjeta Spartan-3E

Para el manejo del Pendubot desde la tarjeta Spartan-3E, se indica en las siguientes secciones.

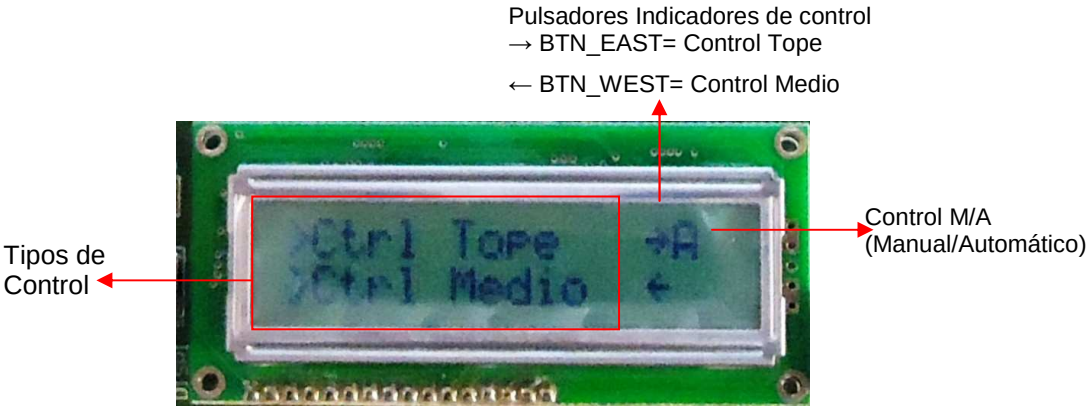


Figura 5.25 Interfaz con Tarjeta Spartan-3E

[1] Instanciar, El crear en memoria un ejemplar de un conjunto de datos y código definido por una clase o estructura

5.5.2.4 Botones perilla, Pulsadores y Switch

Los pulsadores y la perilla permiten seleccionar el tipo de control, encerar los eslabones en las posiciones iniciales, detener el control y variar el set-point.

- BTN_NORTH: Encerar los eslabones
- BTN_SOUTH: Detener el control
- BTN_EAST: Control Tope
- BTN_WEST: Control Medio
- ROT_CENTER: Encerar el Set-point.
- Perilla (ROT_A y ROT_B): Aumenta o disminuye el Set-point

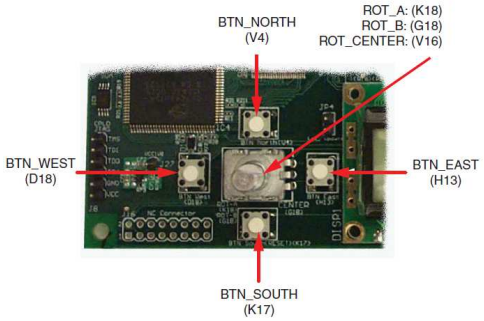


Figura 5.26 Puladores y Perilla [9]

Los Switch permiten iniciar el proceso de control, permiten el control Manual/Automático y mostrar en el LCD el Controlador Utilizado y la posición de los eslabones.

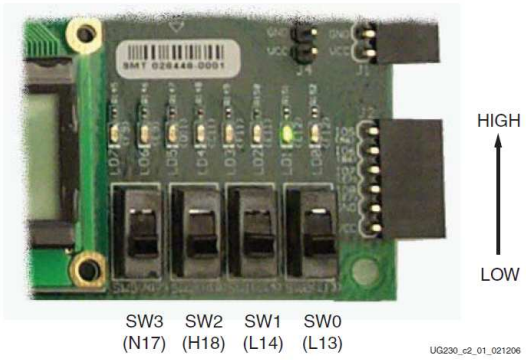


Figura 5.27 Switch [9]

- SW0: Encender Control.
- SW1: Control Manual/Automático.
- SW2: Mostrar posiciones de los eslabones.
- SW3: Mostrar Ganancias del Control LQR.

5.5.2.5 Tipo de control

Se indica en el LCD el tipo de control y las posiciones de los eslabones en grados sexagesimales

Figura 5.27 Control Tope

Figura 5.28 Control Medio

Una vez terminado el programa de control, en el Capítulo 6 se procederá a realizar las pruebas de funcionamiento del Control implementado en FPGA.

CAPÍTULO 6

PRUEBAS Y RESULTADOS

Terminadas las etapas de diseño de los controladores y la lógica del programa para ambas posiciones de control, se realizará en este capítulo las pruebas y resultados experimentales de los controladores en tiempo real, ajuste de los controladores, límites de conmutación y rango de funcionamiento del sistema, usando como sistema de control al FPGA.

La tarjeta Spartan-3E Starter Kit es donde se implementa toda la lógica y los controladores descritos en los capítulos anteriores. Además, se lo utiliza como interfaz de usuario para el manejo del sistema subactuado, ello implica desde la puesta en marcha del sistema, visualización de los parámetros de los controladores e incluyendo el cambio del punto de referencia de funcionamiento del sistema y la comunicación con la PC para una interfaz gráfica con LABVIEW.

6.1 CONFIGURACIÓN PARA COMPILACIÓN

Para poder descargar el sistema de control al FPGA, es necesario compilar el VI.
Al compilar el archivo, LABVIEW FPGA determina el área utilizada en el FPGA, según el número de slices, flip-flops, lookup tables (LUTs), multiplicadores y bloques de memoria RAM utilizados en el diseño, para así conocer el tamaño de implementación en la tarjeta.

Antes de compilar el archivo, es necesario configurar el modo de la FPGA, es decir se debe especificar que el FPGA VI corra o arranque automáticamente cuando se descargue sobre el dispositivo, ya que al no hacerlo no se tendrá independencia con la PC.

La configuración se muestra en la figura 6.1

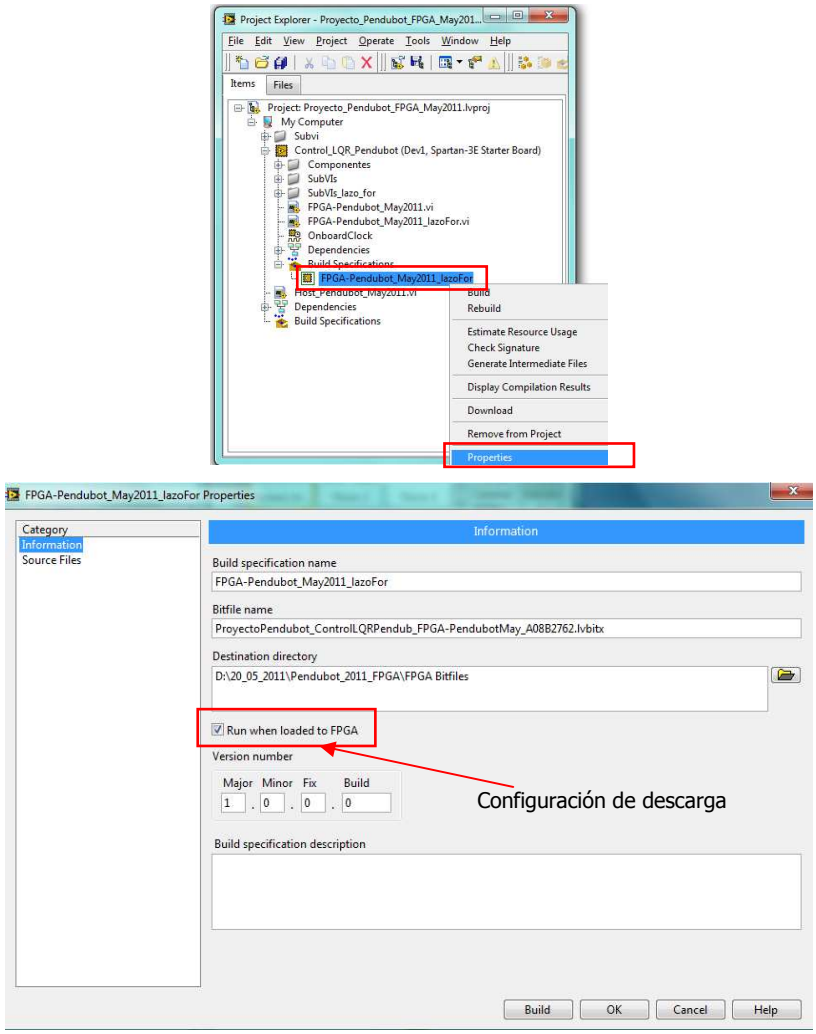


Figura 6.1 Propiedades de configuración del FPGA VI

Los recursos que se han utilizado en el sistema se resumen en la tabla 6.1, este resumen lo realiza el compilador de LabVIEW FPGA.

RECURSOS	DISPONIBLE	UTILIZADOS	PORCENTAJE
Registros Slices	9312	5132	55.1%
Slices	4656	4654	99.9%
LUTs	9312	7784	83.6%
Bloques de RAM	20	1	5.0%
Multiplicaciones	18	13	72.2%

Tabla 6.1 Estimación de Recursos

Se observa que se utiliza una cantidad de recursos menor al máximo disponible, por lo que se puede implementar en la tarjeta Spartan-3E.
Finalizada la compilación, se crean los archivos necesarios para descargar a la tarjeta, El archivo que se descarga a la tarjeta tiene extensión *.lvbitx* y se crea en la carpeta *FPGA Bitfiles* de la carpeta raíz donde se almacena el proyecto principal.

El estado de compilación se muestra en el cuadro de diálogo y cuando se ha visualizado el mensaje de la figura 6.2 "*The compilation complete successfully*", el archivo estará generado.

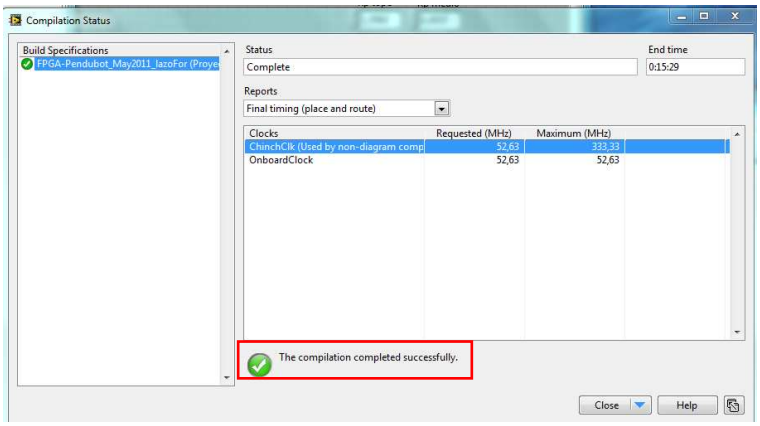


Figura 6.2 Compilación terminada.

6.2 DESCARGA

LABVIEW FPGA permite descargar directamente el archivo a la tarjeta luego de haber terminado la compilación, cabe indicar que se debe descargar el archivo a la memoria Flash de la tarjeta, ya que este proyecto implica que el control se debe realizar sin utilizar la interfaz con la PC, es decir que el programa de control se ejecuta directamente en la FPGA de manera autónoma.

Para ello se selecciona el FPGA VI que se desee descargar, como se muestra en la figura 6.3

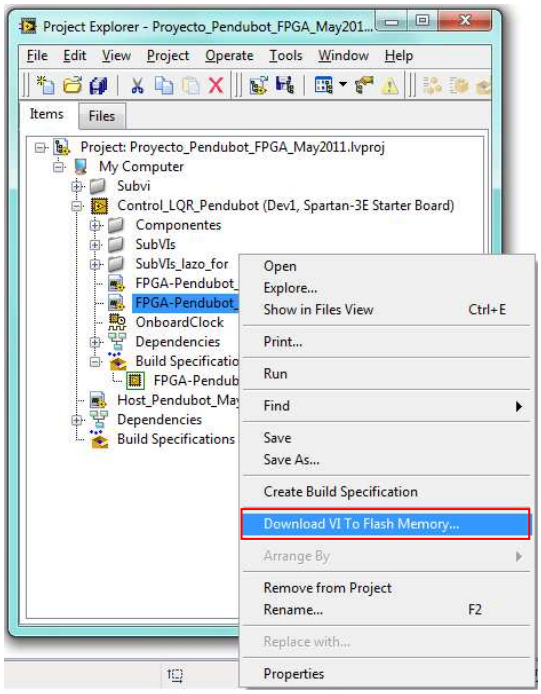


Figura 6.3 Descarga a la tarjeta Spartan-3E Starter Kit.

El mensaje que asegura la descarga del archivo es "*Download Successful*" como se indica en la figura 6.4

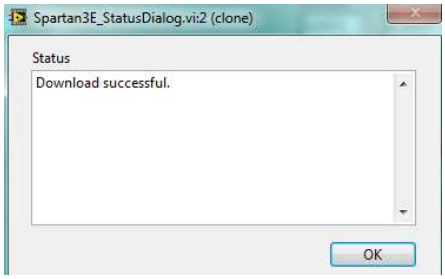


Figura 6.4 Descarga exitosa del archivo .lvbitx a la tarjeta Spartan-3E Starter Kit

Una vez descargado el archivo a la tarjeta, se procede a realizar las pruebas necesarias para el funcionamiento del Pendubot con el FPGA.

6.3 PRUEBAS Y ANÁLISIS DE RESULTADOS.

El funcionamiento adecuado del sistema depende de cada una de las partes integrantes del Pendubot, siendo necesario realizar algunas pruebas que permitan verificar el correcto desempeño de cada una de ellas.

6.3.1 FUNCIONAMIENTO DE LA TARJETA DE MANEJO DE ENCODERS Y DE AJUSTE DE SEÑAL DE CONTROL

Los codificadores ópticos utilizados para sensar las posiciones angulares de los eslabones entregan una señal de +5V como 1L, para lo cual se emplea un circuito adicional que entregue un voltaje de 3.3V como 1L (estándar LVTTL), para no ocasionar daños a la tarjeta. (Sección 5.1 Adquisición de Señales)

Además en la misma placa se ha situado el acondicionamiento de la señal de control para obtener la señal adecuada (Sección 5.3 Salida de Señales)

Se realizan las pruebas necesarias para comprobar que los ajustes fueron los adecuados y que se obtienen los voltajes necesarios para que el sistema funcione correctamente.

6.3.2 FUNCIONAMIENTO DE LOS ALGORITMOS DE CONTROL

Una vez verificado el desempeño adecuado del hardware, se procede a la comprobación de los módulos del control y de los algoritmos de control implementados en la FPGA.

Para ello se probaron los algoritmos de control por separado, con la finalidad de observar el comportamiento individual sobre el sistema y los efectos de la señal de control.

Considerando diferentes puntos de operación (en el control de equilibrio) y variando los diferentes controladores, se procedió a experimentar con diferentes ganancias, lo cual permitirá la verificación de los algoritmos de control, permitiendo además establecer los rangos de operación de la posición angular de los enlaces alrededor de los puntos de equilibrio.

6.3.3 RESULTADOS

Verificado el funcionamiento de cada controlador por separado, se procede a realizar el control del Pendubot con la FPGA en las posiciones tope y medio,

luego utilizando el control híbrido, es decir el control de balanceo y el control de equilibrio para cada una de las posiciones.

En esta sección se muestra que con la llegada de los FPGA se tienen más beneficios a parte del desarrollo rápido y de bajo costo. Los FPGA pueden ser reconfigurados una y otra vez, se puede diseñar hardware que pueden ser actualizados en el campo, que las FPGA han incrementado la vida útil y la confiabilidad de los sistemas ya que se aprovecha las ventajas de su capacidad de reprogramación.

Y se comprobará que se puede configurar un complejo sistema de computador como es desde la adquisición de datos hasta la interfaz de usuario para un sistema de control en un solo circuito, en donde se ha implementado los dos tipos de controles.

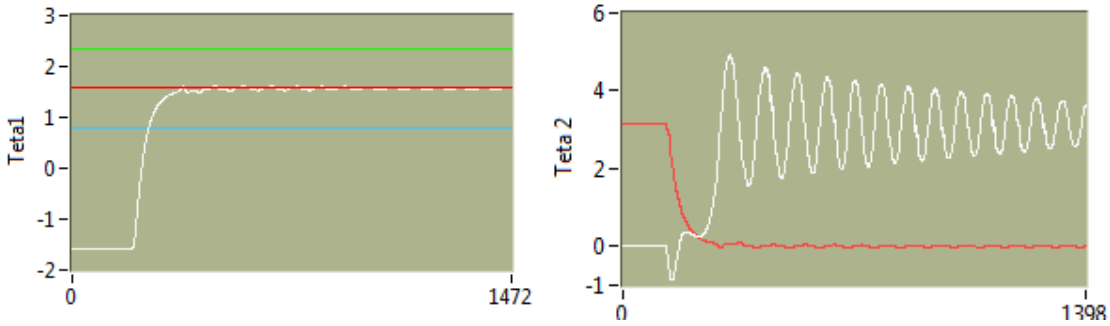
Para realizar las pruebas con ambos controladores es necesario que el Sistema se encuentre en la posición inicial, dependiendo si es control manual o automático.

- Control Manual: la posición inicial debe ser de acuerdo al tipo de control. Si se realiza el control Tope la posición inicial debe ser alrededor del punto $(\frac{\pi}{2}, 0)$ y en el control Medio la posición inicial debe ser alrededor del punto $(-\frac{\pi}{2}, \pi)$.
- Control Automático: la posición inicial es igual para ambos tipos de control $(-\frac{\pi}{2}, 0)$ (posición estable de equilibrio)

El funcionamiento del control puede observarse al analizar las curvas de posición de los enlaces, realizando la conexión entre la FPGA y la computadora.

6.3.3.1 Control de Balanceo y Equilibrio del Pendubot en la Posición Tope

A continuación se implementa el algoritmo de control para el balanceo y se muestran las gráficas de las respuestas para ambos eslabones, figura 6.5, partiendo desde la posición de equilibrio inferior y desde ahí el sistema evoluciona hasta llegar a los alrededores del punto de equilibrio inestable para ambos eslabones.



Eslabón 1 Eslabón 2

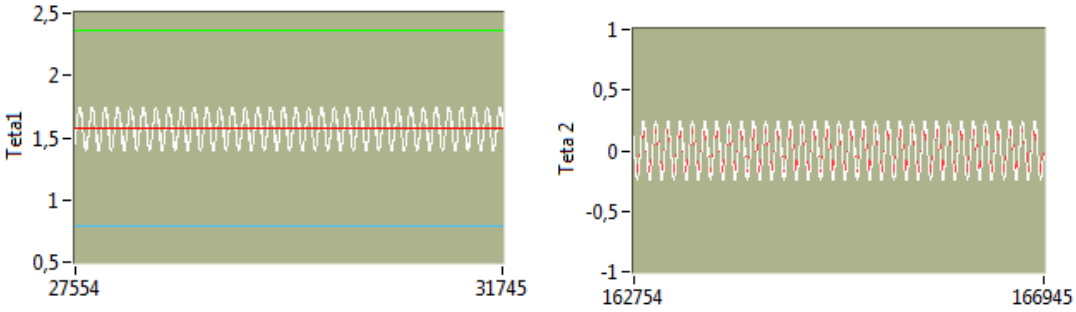
Figura 6.5 Control Balanceo Posición Tope

Esto es lo que se logra con el control PD con linealización parcial. La referencia de posición es un escalón de $\frac{\pi}{2}$ para el eslabón linealizado (eslabón 1). Como se puede observar el segundo eslabón se mantiene durante un instante de tiempo en la posición deseada para la posición tope (en donde se puede conmutar a la segunda ley de control) y luego oscila libremente siguiendo su propia dinámica. Las ganancias del control PD, se obtuvieron a prueba y error (apartado 3.1.2.2.1).

$$k_p = 1{,}0282$$
$$k_d = 0{,}0527$$

Como se puede observar el control de balanceo del Pendubot es resuelto en forma satisfactoria.

Seguidamente se implementa el algoritmo de control para el equilibrio y se muestran las gráficas de las respuestas para ambos eslabones, figura 6.6, partiendo desde la posición de equilibrio inestable superior, donde se equilibra el sistema en esa posición.



Eslabón 1 Eslabón 2

Figura 6.6 Control Equilibrio Posición Tope

Utilizando las siguientes Matrices de ponderación:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad R = [1,1]$$

El vector óptimo de control es:

$$K_{Tope} = \begin{bmatrix} -10.7180 \\ -10.1647 \\ -2.1136 \\ -1.4222 \end{bmatrix}$$

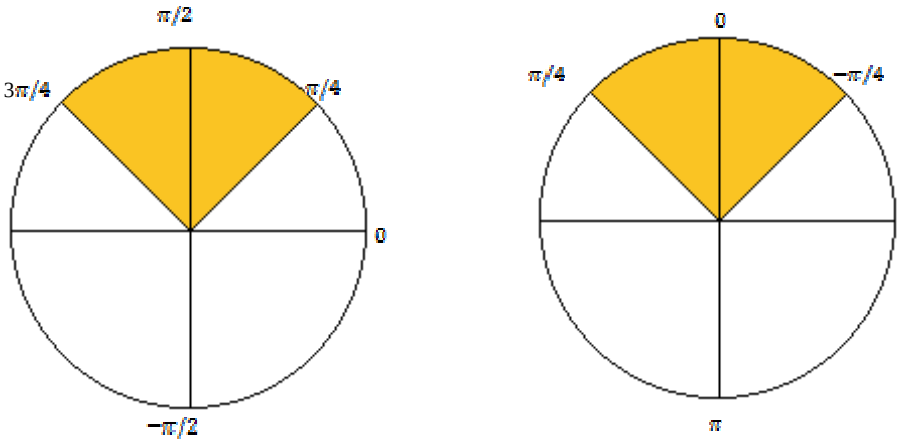
Se debe recalcar que el rango de control en la posición tope se ha limitado debido a que el sistema está linealizado a un solo punto de equilibrio. De esta manera el rango de variación para el eslabón 1 está dada por:

$$\frac{\pi}{4}\ rad < \theta_1 < \frac{3\pi}{4}\ rad$$

Esto también permite variar el punto de operación dentro de este rango de control.

El movimiento del eslabón 2 es controlado por el movimiento del eslabón 1 a través de los algoritmos de control. Con esto la variación del eslabón 2 está dada por la relación: $\theta_1 = \frac{\pi}{2} - \theta_2$, que da un rango de:

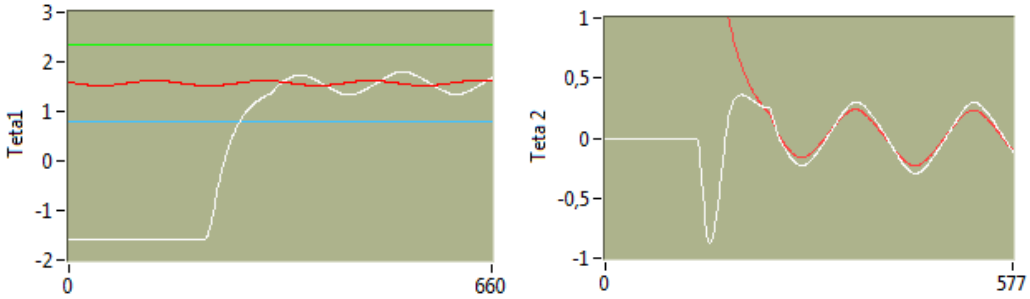
$$\frac{\pi}{4}\ rad < \theta_2 < -\frac{\pi}{4}\ rad$$



Eslabón 1 Eslabón 2

Figura 6.7 Límites de Control del Pendubot en posición Tope

En seguida se muestra la respuesta del Pendubot al aplicarle la ley de control que resuelve el problema de balanceo y de equilibrio, es decir el control híbrido del sistema. El control inicial es el control de Balanceo y cuando el sistema está cerca del punto de equilibrio inestable superior conmuta al segundo control, la ley de control de equilibrio. En la siguiente figura se muestra el funcionamiento completo, en ambas gráficas el sistema inició desde la posición inferior (la posición estable de equilibrio).



Eslabón 1 Eslabón 2

Figura 6.8 Control Balanceo y Equilibrio de la Posición Tope

La figura 6.9 muestra el Control Tope del Pendubot.

Figura 6.9 Control en la posición Tope

Como se esperaba, el algoritmo de control lleva los enlaces desde la posición estable hasta la posición inestable de equilibrio por medio del control de balanceo y lo mantiene en esa posición utilizando el controlador de equilibrio.

Las ganancias del vector óptimo se pueden reajustar a través de la interfaz con el computador en tiempo real, con la finalidad de observar el comportamiento del sistema a diferentes ganancias del vector óptimo de control.

A continuación se muestran diferentes comportamientos del sistema al variar el controlador en la posición tope y su punto de operación.

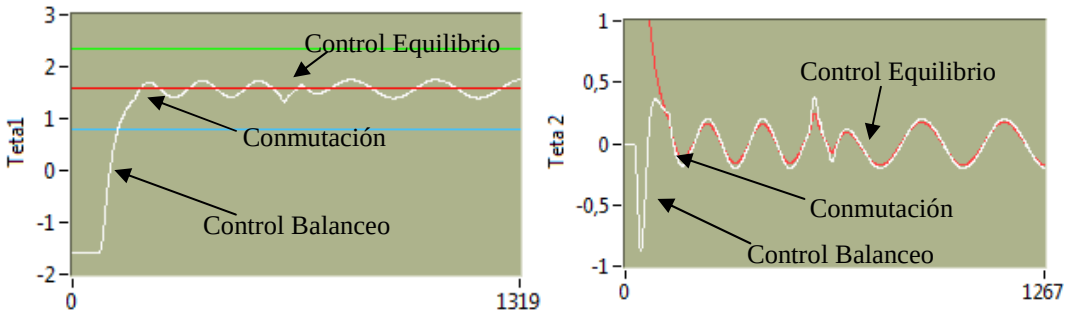
6.3.3.1.1 Comportamiento desde la posición estable de equilibrio (posición inferior)

Se realizarán diferentes pruebas del sistema con la finalidad de observar el comportamiento del sistema ante diferentes controladores del vector óptimo, las pruebas se realizarán a partir del punto de equilibrio estable (posición inferior).

- Para: $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3.50 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ y $R = [1.1]$, se tiene:

$$K_{Tope} = \begin{bmatrix} -13.7134 \\ -12.6425 \\ -2.6928 \\ -1.7778 \end{bmatrix}$$

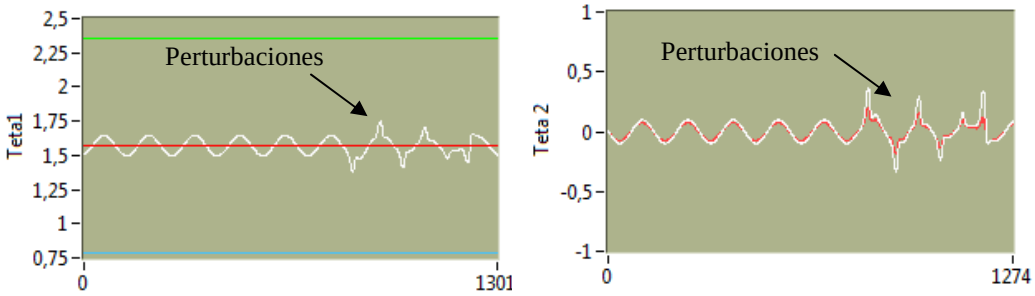
Las figuras del comportamiento correspondiente se muestran a continuación.



Eslabón 1 Eslabón 2

Figura 6.10 Posiciones angulares

También se puede observar el efecto ante perturbaciones en el sistema.



Eslabón 1 Eslabón 2

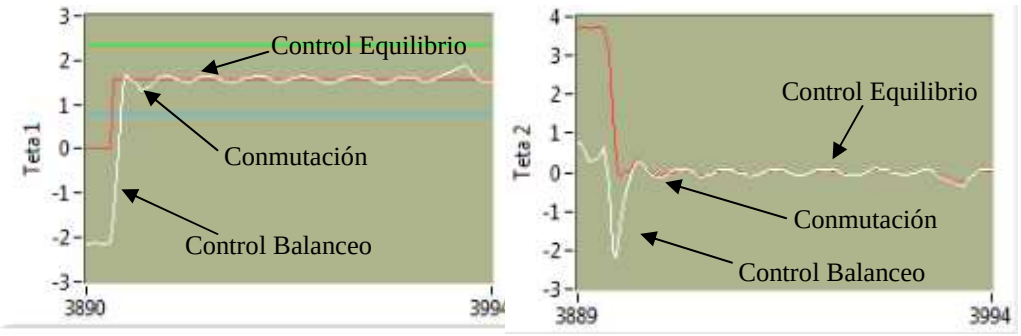
Figura 6.11 Posición angular de los eslabones ante perturbaciones

Para este controlador se puede observar que la variación de la posición de ambos enlaces oscila alrededor del punto de equilibrio deseado, aproximadamente en 0.17 rad (9°).

- Para: $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 24.5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ y $R = [1.1]$, se tiene:

$$K_{Tope} = \begin{bmatrix} -26.4052 \\ -23.0227 \\ -5.1425 \\ -3.2687 \end{bmatrix}$$

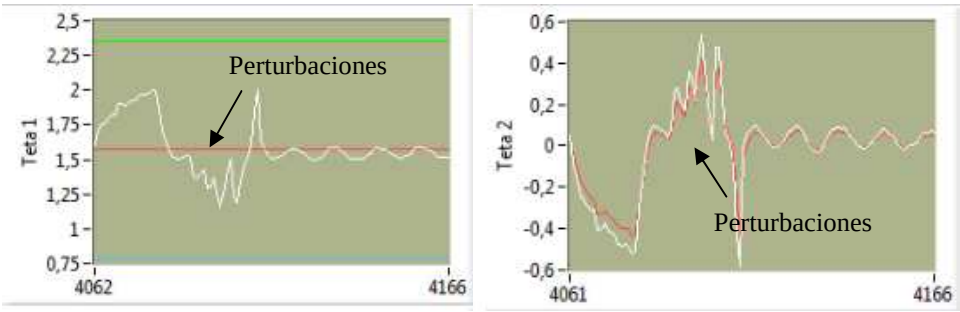
Las figuras del comportamiento correspondiente se muestran a continuación.



Eslabón 1 Eslabón 2

Figura 6.12 Posiciones angulares

También se puede observar el efecto ante perturbaciones en el sistema.



Eslabón 1 Eslabón 2

Figura 6.13 Posición angular de los eslabones ante perturbaciones

Para este controlador se puede observar que la variación de la posición de ambos enlaces tiene una oscilación menor alrededor del punto de equilibrio deseado, aproximadamente en 0.09 rad (5.15°).

El Control de Balanceo (Control PD) es el mismo para cualquier cambio del controlador de equilibrio

Como se esperaba el algoritmo de control de llevar los enlaces desde la posición estable de equilibrio (posición inferior) hasta la posición inestable de equilibrio (posición tope) se cumple siempre y cuando el sistema esté en la posición inicial; y, el algoritmo de control permite estabilizar al sistema en la posición deseada, además se cumple la condición de $\theta_1 + \theta_2 = \frac{\pi}{2}$, lo cual se verifica si se suman las señales de las posiciones de cada eslabón.

6.3.3.1.2 Comportamiento desde la posición inestable de equilibrio (posición tope)

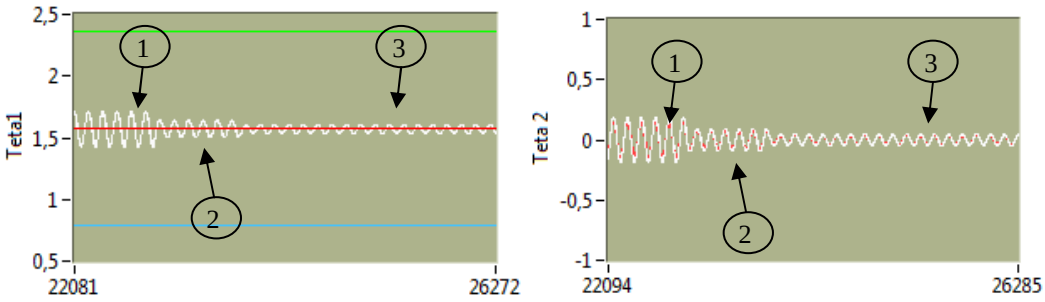
Estas pruebas se realizarán con el sistema en la posición tope, en donde se probará la rapidez de procesamiento del algoritmo de control implementado en la FPGA, al cambiar el controlador de equilibrio cuando el sistema está en funcionamiento.

- Se han probado los siguientes controladores para observar el comportamiento.

1. $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 3.50 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $R = [1.1]$, $K_{Tope} = \begin{bmatrix} -13.7134 \\ -12.6425 \\ -2.6928 \\ -1.7778 \end{bmatrix}$

2. $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $R = [1.1]$, $K_{Tope} = \begin{bmatrix} -18.2538 \\ -16.3726 \\ -3.5696 \\ -2.3134 \end{bmatrix}$

3. $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 24.50 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $R = [1.1]$, $K_{Tope} = \begin{bmatrix} -26.4052 \\ -23.0227 \\ -5.1425 \\ -3.2687 \end{bmatrix}$



Eslabón 1 Eslabón 2

Figura 6.14 Posición angular de los eslabones ante cambios del controlador

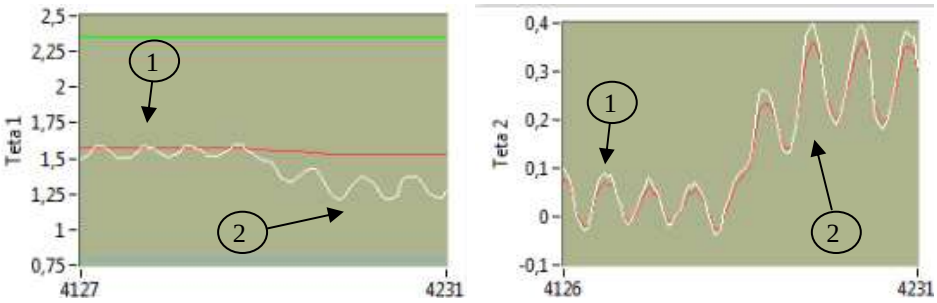
Al equilibrar al Pendubot en la Posición Tope, se observa que existe una oscilación alrededor del punto de equilibrio, donde la amplitud depende del controlador utilizado. Al investigar esto se han realizado varias mediciones sobre el motor y el servoamplificador. En el servoamplificador se encontró que éste amplifica las tensiones positivas más que las tensiones negativas. La amplificación debe ser uniforme para tener un mejor resultado. Y en el motor al tener 40 delgas dispuestas sobre el eje del rotor, implica que se tiene un cambio entre delga y delga de 9°, podría influir directam ente en el control fino que se requeriría aplicar, además existe diferentes características en cada sentido de giro del motor, es decir que en sentido anti-horario el motor presenta mayor velocidad que en sentido horario. Estas características se cree que es la causa de las oscilaciones observadas, además de la alta no linealidad del sistema Pendubot.

6.3.3.1.3 Comportamiento al variar el punto de operación del sistema.

Una vez equilibrado el sistema, se variará el punto de operación, para probar la robustez del control en plantas no lineales, como lo es el Pendubot.

- Para los siguientes puntos de referencia:
    1. Punto de Referencia inicial = 1.5708 rad
    2. Punto de Referencia final = 1.5347 rad

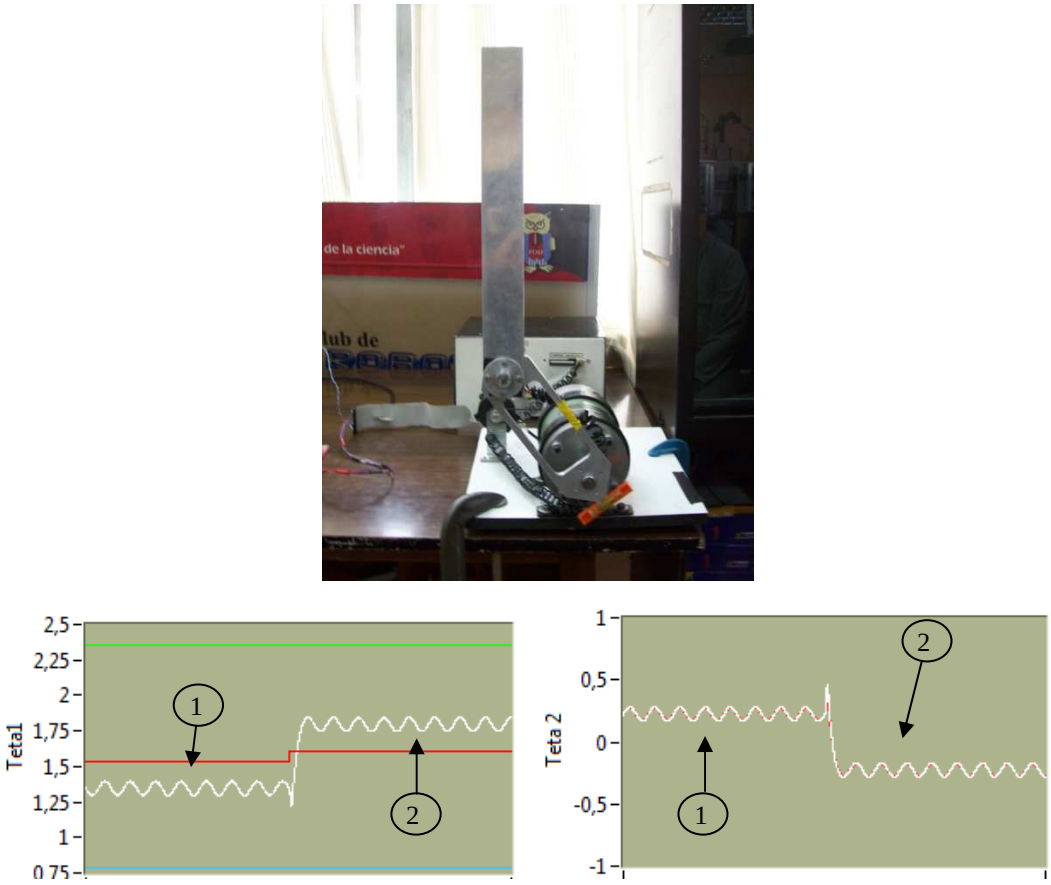
(Continúa)



Eslabón 1 Eslabón 2

Figura 6.15 Posición angular de los eslabones ante cambios del punto de referencia

- Para los siguientes puntos de referencia:
    1. Punto de Referencia inicial = 1. 5347 rad
    2. Punto de Referencia final = 1.6068 rad



Eslabón 1 Eslabón 2

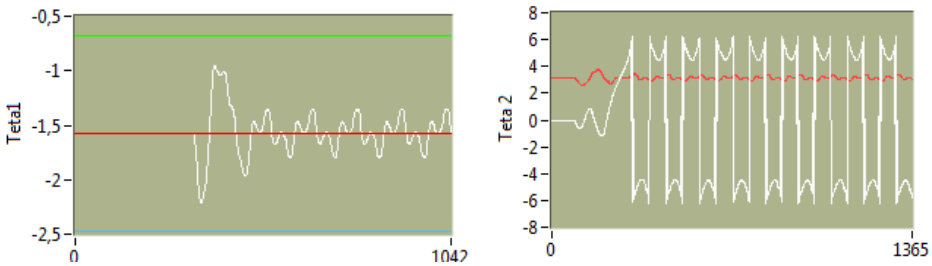
Figura 6.16 Posición angular de los eslabones ante cambios del punto de referencia

Equilibrar al Pendubot en otro punto de referencia es posible con ambos eslabones en la posición Tope, se observa que también existe una oscilación alrededor del punto equilibrado. Como se esperaba el algoritmo de control permite estabilizar al sistema en la posición deseada, cumpliéndose siempre la condición de $\theta_1 + \theta_2 = \frac{\pi}{2}$.

6.3.3.2 Control de Balanceo y Equilibrio del Pendubot en la Posición Media

Como en el caso anterior se muestra el control del Pendubot en la posición media. Se prueba el algoritmo de control para el balanceo, figura 6.17, partiendo desde la posición de equilibrio inferior y desde ahí el sistema evoluciona hasta llegar a los alrededores del punto de equilibrio inestable para ambos eslabones.

Como se indicó en 3.1.2.2.2 es necesaria una señal inicial de referencia la cual es de -0,94 V que se aplica durante 0,5948 segundos, para incrementar la energía del sistema.



Eslabón 1 Eslabón 2

Figura 6.17 Control Balanceo Posición Medio

Al igual que en el control anterior, se puede observar que el segundo eslabón se mantiene durante un instante de tiempo en la posición deseada, posición media (en donde se puede conmutar a la segunda ley de control), y luego oscila libremente siguiendo su propia dinámica. La referencia para esta posición es un escalón de - $\frac{\pi}{2}$ para el eslabón linealizado (eslabón 1).

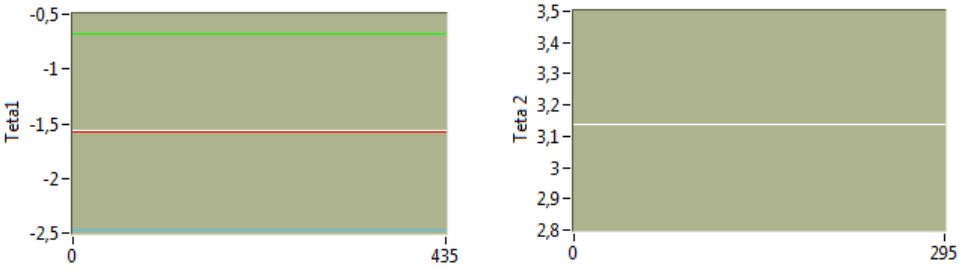
Las ganancias del control PD para la Posición Media, se obtuvieron a prueba y error (apartado 3.1.2.2.2).

$$k_p = 0{,}9457$$

$$k_d = 0{,}0359$$

Como se puede observar el control de balanceo para la Posición Media del Pendubot también es resuelto de forma satisfactoria.

Seguidamente se implementa el algoritmo de control de equilibrio y se muestran las gráficas de las respuestas para ambos eslabones, figura 6.18, partiendo desde la posición de equilibrio inestable medio, donde se equilibra el sistema en esa posición.



Eslabón 1 Eslabón 2

Figura 6.18 Control Equilibrio Posición Medio

Utilizando las siguientes Matrices de ponderación:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad R = [1{,}1]$$

El vector de control óptimo es:

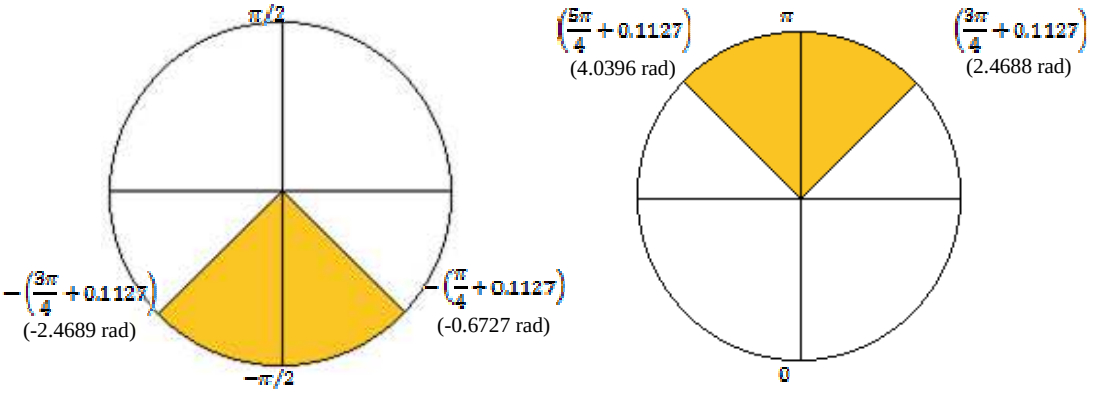
$$K_{Medio} = \begin{bmatrix} 4.8134 \\ 7.3641 \\ 0.4950 \\ 1.0054 \end{bmatrix}$$

Se debe recalcar que el rango de control en la Posición Media también se limitó. De esta manera el rango de variación para el eslabón 1 está dada por:

$$-\left(\frac{3\pi}{4} + 0.1127\right) rad < \theta_1 < -\left(\frac{\pi}{4} + 0.1127\right) rad$$

El movimiento del eslabón 2 es controlado por el movimiento del eslabón 1 a través de los algoritmos de control. Con esto la variación del eslabón 2 está dada por la relación: $\theta_1 = \frac{\pi}{2} - \theta_2$, que da un rango de:

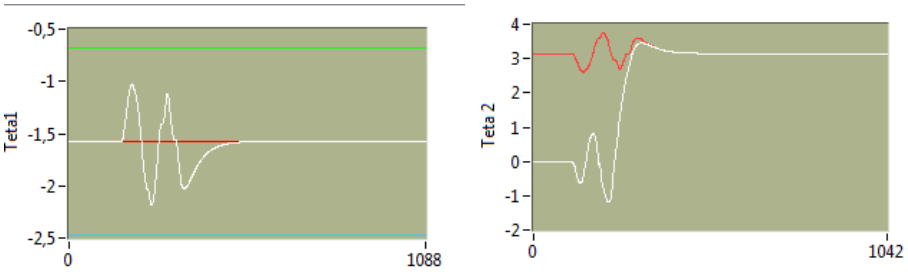
$$\left(\frac{5\pi}{4} + 0.1127\right) rad < \theta_2 < \left(\frac{3\pi}{4} + 0.1127\right) rad$$



Eslabón 1 Eslabón 2

Figura 6.19 Límites de Control del Pendubot en Posición Media.

A continuación se muestra la respuesta del Pendubot al aplicarle la ley de control que resuelve el problema de balanceo y de equilibrio, es decir el control híbrido del sistema. En la siguiente figura se muestra el modo de funcionamiento, en ambas gráficas el sistema inició desde la posición inferior (la posición estable de equilibrio).



Eslabón 1 Eslabón 2

Figura 6.20 Control Balanceo y Equilibrio de la Posición Medio

La figura 6.21 muestra el Control Medio del Pendubot.

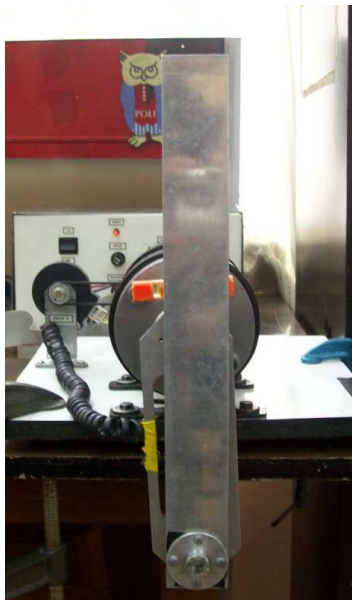


Figura 6.21 Control en la Posición Media

Al igual que el caso anterior, el algoritmo de control lleva los enlaces desde la posición estable hasta la posición inestable de equilibrio por medio del control de balanceo y lo mantiene en esa posición utilizando el controlador de equilibrio.

Las ganancias del vector óptimo para esta posición también se pueden reajustar o cambiar a través de la interfaz con el computador en tiempo real, con la finalidad

de observar el comportamiento del sistema a diferentes ganancias del vector óptimo de control.

A continuación se muestran diferentes comportamientos del sistema al variar el controlador para la posición media.

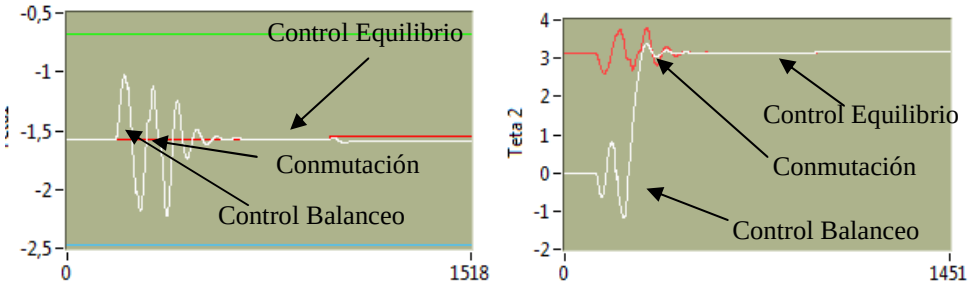
6.3.3.2.1 Comportamiento desde la posición estable de equilibrio (posición inferior)

Todas las pruebas y consideraciones realizadas para la posición tope han sido aplicadas de igual manera para la Posición Media.

- Para: $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ y $R = [1.1]$, se tiene:

$$K_{Medio} = \begin{bmatrix} 4.8134 \\ 7.3641 \\ 0.4950 \\ 1.0054 \end{bmatrix}$$

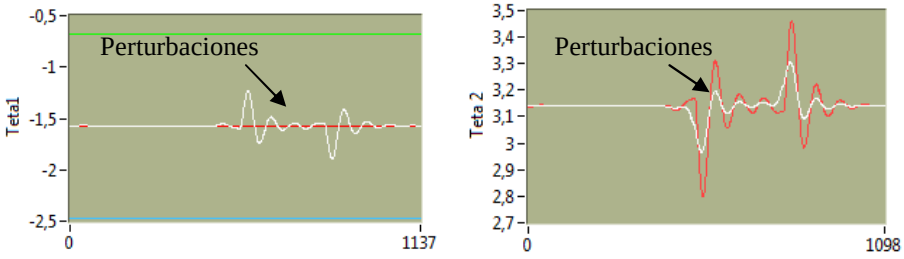
Las figuras del comportamiento correspondiente se muestran a continuación.



Eslabón 1 Eslabón 2

Figura 6.22 Posiciones angulares

También se puede observar el efecto ante perturbaciones en el sistema.



Eslabón 1 Eslabón 2

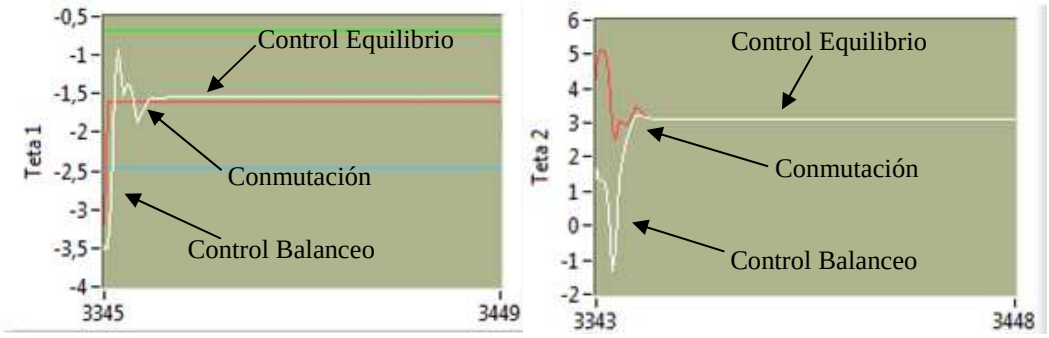
Figura 6.23 Posición angular de los eslabones ante perturbaciones

Para este controlador se puede observar que la variación de la posición de ambos enlaces esta alrededor del punto de equilibrio deseado, aproximadamente en 0.03 rad (1.72°).

- Para: $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$ y $R = [1.1]$, se tiene:

$$K_{Medio} = \begin{bmatrix} 7.5478 \\ 10.2754 \\ 0.7464 \\ 1.4329 \end{bmatrix}$$

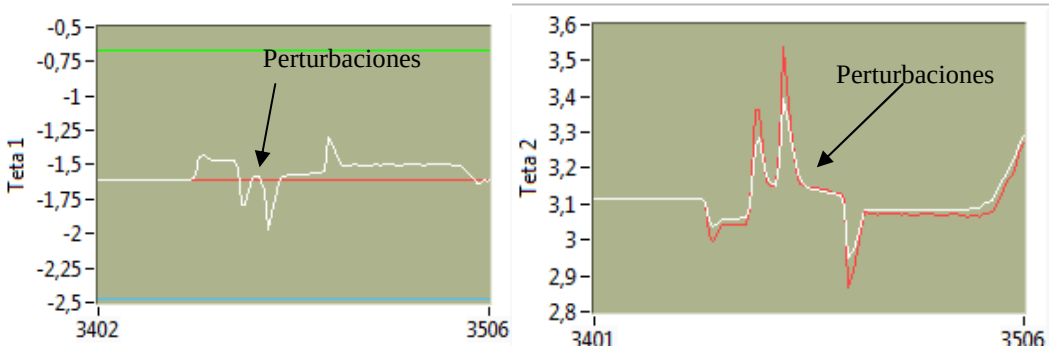
Las figuras del comportamiento correspondiente se muestran a continuación.



Eslabón 1 Eslabón 2

Figura 6.24 Posiciones angulares

También se puede observar el efecto ante perturbaciones en el sistema.



Eslabón 1 Eslabón 2

Figura 6.25 Posición angular de los eslabones ante perturbaciones

Para este controlador se puede observar que la variación de la posición de ambos enlaces esta alrededor del punto de equilibrio deseado, aproximadamente en 0.03 rad (1.72°).

El Control de Balanceo (Control PD) es el mismo para cualquier cambio del controlador de equilibrio

Como se esperaba el algoritmo de control de llevar los enlaces desde la posición estable de equilibrio (posición inferior) hasta la posición inestable de equilibrio (posición media) se cumple siempre y cuando el sistema esté en la posición inicial; y, el algoritmo de control permite estabilizar al sistema en la posición deseada, además que se cumple la condición de $\theta_1 + \theta_2 = \frac{\pi}{2}$, lo cual se verifica si se suman las señales de las posiciones de cada eslabón.

6.3.3.2.2 Comportamiento desde la posición inestable de equilibrio (posición media)

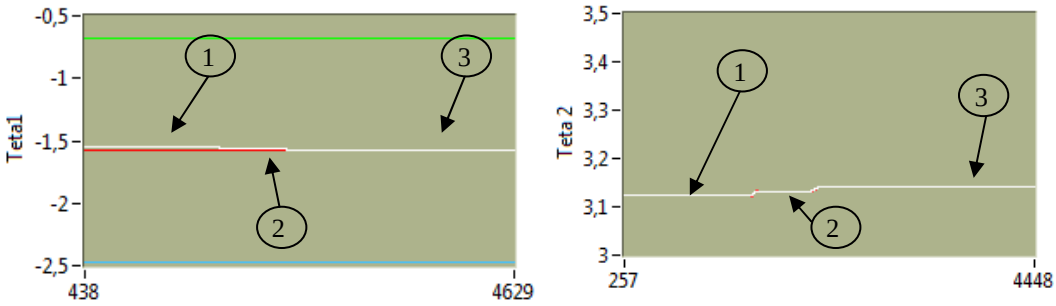
Estas pruebas se realizarán con el sistema en la posición media, en donde se probará la rapidez de procesamiento del algoritmo de control implementado en la FPGA, al cambiar el controlador de equilibrio cuando el sistema está en este punto de operación.

- Se han probado los siguientes controladores para observar el comportamiento.

1. $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $R = [1.1]$, $K_{Medio} = \begin{bmatrix} 4.8134 \\ 7.3641 \\ 0.4950 \\ 1.0054 \end{bmatrix}$

2. $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $R = [1.1]$, $K_{Medio} = \begin{bmatrix} 7.5478 \\ 10.2754 \\ 0.7464 \\ 1.4329 \end{bmatrix}$

3. $Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 4.5 & 0 & 0 \\ 0 & 0 & 0.05 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$, $R = [1.1]$, $K_{Medio} = \begin{bmatrix} 8.2517 \\ 11.4514 \\ 0.8016 \\ 1.595 \end{bmatrix}$



Eslabón 1 Eslabón 2

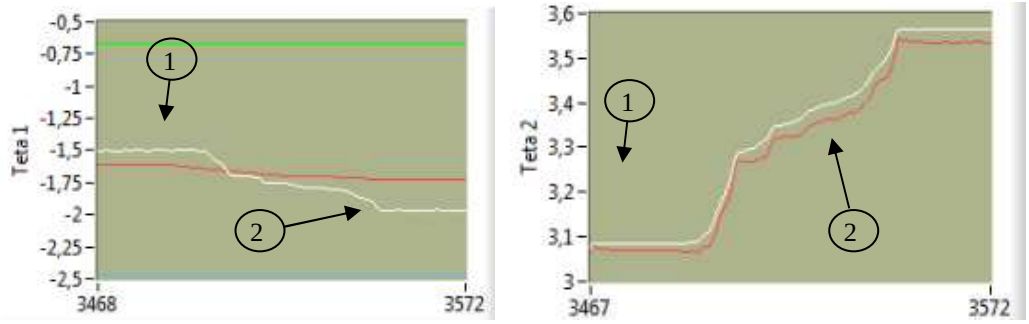
Figura 6.26 Posición angular de los eslabones ante cambios del controlador

Al equilibrar al Pendubot en la posición media, se observa que esta posición es más estable que la posición tope, por lo que permite un mejor control del Pendubot.

6.3.3.2.3 Comportamiento al variar el punto de operación del sistema.

Una vez equilibrado el sistema, se variará el punto de operación, para probar la robustez del control en plantas no lineales como el Pendubot

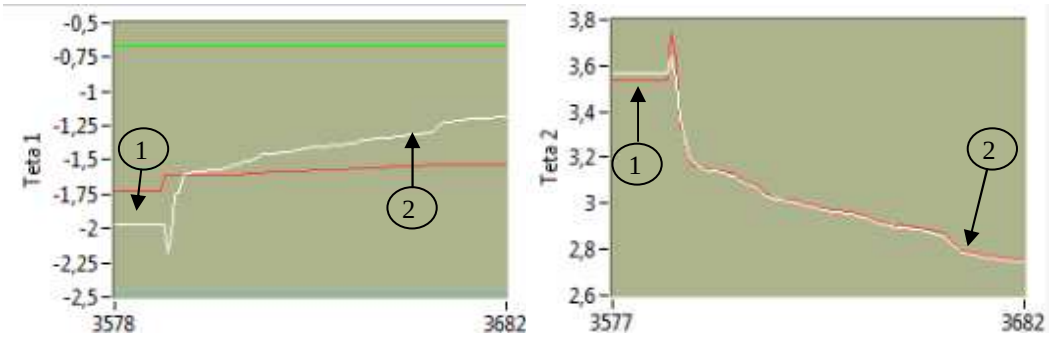
- Para los siguientes puntos de referencia:
    1. Punto de Referencia inicial = -1.5708 rad
    2. Punto de Referencia final = -1.6626 rad



Eslabón 1 Eslabón 2

Figura 6.27 Posición angular de los eslabones ante cambios del punto de referencia

- Para los siguientes puntos de referencia:
  1. Punto de Referencia inicial = -1.6626rad
  2. Punto de Referencia final = -1.4595 rad



Eslabón 1 Eslabón 2

Figura 6.16 Posición angular de los eslabones ante cambios del punto de referencia

Equilibrar al Pendubot en otro punto de referencia es posible con ambos eslabones en la posición media. Como se esperaba el algoritmo de control permite estabilizar al sistema en la posición deseada, cumpliéndose siempre la condición de $\theta_1 + \theta_2 = \frac{\pi}{2}$.

Los resultados presentados muestran un funcionamiento adecuado del sistema, en las posiciones tope y media a través del FPGA.

Los módulos programados que se emplean para el algoritmo de este proyecto, demuestran la gran versatilidad de la programación en alto nivel sobre los FPGA, debido a que se puede realizar o alterar el programa principal de una manera fácil.

La adopción de la tecnología FPGA continúa creciendo mientras que las herramientas de alto nivel evolucionan para ofrecer mejores beneficios del silicio reprogramable.

CAPÍTULO 7

CONCLUSIONES Y RECOMENDACIONES

Se desarrolló una metodología que permita controlar (llevar a la posición de equilibrio y manejar cualquier perturbación) de manera óptima y en tiempo real al Pendubot empleando técnicas de control moderno y clásico, a través de una FPGA

7.1 CONCLUSIONES

La adaptación del Módulo Pendubot para ser controlado por una FPGA y el diseño de los diferentes controladores (controlador Clásico para el Control del Balanceo y un controlador Moderno para el Control de Equilibrio) que conforman el algoritmo de control, han permitido el correcto funcionamiento del sistema, cumpliéndose de esta manera con el objetivo del trabajo propuesto, que es implementar algunas técnicas de control sobre la FPGA utilizando LabVIEW como plataforma de desarrollo.

El algoritmo de control desarrollado para el Control de Balanceo se diseñó mediante la linealización por realimentación parcial, donde una vez linealizado se aplicó un control PD (control clásico), el cual permitió llevar al sistema alrededor del punto de operación; y, el algoritmo de control desarrollado para el Control de Equilibrio del sistema se diseñó mediante la técnica del LQR (control moderno), el cual permitió estabilizar al Pendubot en las configuraciones tope y media de equilibrio, mostrando su eficacia y robustez al momento de estabilizar al sistema y ante cualquier perturbación existente, además se diseño una interfaz donde se puede variar la matriz óptima de control *K* y así comprobar diferentes matrices óptimas de control y poder observar el comportamiento del sistema al cambiarlo en tiempo real. Inclusive la interfaz permite variar el punto de operación del Pendubot, ampliando así el desempeño de los controladores sobre un sistema altamente no lineal.

Al realizar las pruebas sobre el Pendubot, se observó que el sistema es incapaz de llegar a las posiciones deseadas, para superar este inconveniente fue necesario agregar una oscilación al sistema en lazo abierto en dirección contraria al movimiento con el fin de que el sistema gane inercia y luego conmutar a la ley de control de balanceo el cual permitió llevar a la posición deseada.

El estudio de técnicas hibridas de control son una herramienta útil para el manejo de sistemas no lineales. Estas permiten hacer una descripción del sistema y establecer las regiones de operación de cada controlador, con el fin de tomar una decisión para ejecutar cada estrategia de control.

Los rangos de operación han sido ampliados para poder variar el punto de operación del sistema y así poder observar la robustez del controlador dentro de un mayor rango.

Los resultados obtenidos al realizar el control en la posición tope, muestra una oscilación permanente alrededor del punto de equilibrio, debido a las imperfecciones del servoamplificador y a las características del motor y a la fuerte no linealidad del Pendubot, pues al linealizar el modelo solo es válido en su punto de operación.

Es de anotar que durante las pruebas ninguna de las técnicas de control seleccionadas eran útiles cuando se tenían condiciones alejadas a las iniciales. Ello debido a que las oscilaciones y la complejidad del sistema no coincidían con la condición inicial del controlador.

La limitación principal en la implementación del algoritmo, lo constituyó la memoria del FPGA debido a que LabVIEW por ser un lenguaje de alto nivel genera lo que comúnmente se llama “código basura”, este problema se ha solucionado al reducir y optimizar el código, de manera que los operadores matemáticos (específicamente los multiplicadores) sean reutilizados, al incluir los lazos *for* dentro de los módulos del algoritmo.

La ventaja principal de programar en LabVIEW (lenguaje de alto nivel) es que permite programar a la FPGA sin tener conocimiento del HDL (Lenguaje de Descripción de Hardware), además que permite resolver el problema de las operaciones de coma flotante, facilitando así la programación, ya que al implementar estas operaciones en un lenguaje HDL resulta bastante complejo.

Los resultados experimentales validan a la FPGA para realizar control de sistemas, obteniéndose un buen funcionamiento del algoritmo de control y adquisición, tanto a la facilidad de realizar cambios en el diseño como también realizar múltiples procesos.

7.2 RECOMENDACIONES

Se deja para desarrollos futuros la aplicación de otras técnicas de control como la lógica difusa, la identificación con redes neuronales y control con visión artificial para el control de este sistema utilizando el potencial del FPGA programando en Lenguaje HDL, además de poder realizar el cálculo de la matriz de ganancia *K* en el mismo FPGA y así tener independencia total del sistema con un computador, para ello se debería utilizar una tarjeta con mayor memoria para ser programado en LabVIEW.

Debido a que los dispositivos utilizados son muy sensibles, específicamente el FPGA y encoders, es aconsejable tomar las precauciones necesarias para evitar daños permanentes en los dispositivos. Para ello se recomienda verificar las conexiones en el manual de usuario (Anexo A).

REFERENCIAS BIBLIOGRÁFICAS

[1] WIKIPEDIA, http://es.wikipedia.org/wiki/FPGA

[2] D. Block D. and M. Spong, “Mechanical design and control of the pendubot,” in 46th *Annual Earthmoving Industry Conference*, SAE technical paper series, 1995.

[3] Andrade S. Renato German, “Análisis, Diseño y Construcción del PENDUBOT”. TESIS DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA, EPN, 2000

[4] Starkov Konstantin Ing, “Control de Movimiento en Sistemas Mecánicos Subactuados”. Maestría en Ciencias Digitales, Instituto Politécnico Nacional, México, 2008

[5] SLOTINE Jean-Jacques, and LI Weiping, Applied Nonlinear Control, Prentice Hall, New Jersey 1991.

[6] Rojas P. Adolfo, “ANÁLISIS E IMPLEMENTACIÓN DE LA SIMULACIÓN EN TIEMPO REAL DE UN SISTEMA SUBACTUADO PENDUBOT”, Maestría en Tecnología Avanzada, Instituto Politécnico Nacional, México, 2007

[7] KALIPEDIA, http://www.kalipedia.com/tecnologia/tema/graficos-encoder-optico.html?x1=20070821klpinginf_48.Ees&x=20070821klpinginf_92.Kes&x2=20070821klpinginf_89.Kes

[8] National Instruments, Notas de aplicación, “CONVERSIÓN DE SEÑAL DE ENCODER DE CUADRATURA EN SEÑAL DE PULSOS Y DERECHA-IZQUIERDA”, www.highlights.com.ec, Cuenca-Ecuador.

[9] Spartan-3E FPGA Starter Kit Board User Guide, Xilinx UG230 (v1.1) June 20, 2008.

[10] Cruz V. Carlos A, Gallegos Alvarez, Villarreal C. Miguel, “Rediseño Paramétrico del Pendubot para posicionamiento vertical en tiempo mínimo”, Instituto Politécnico Nacional, México, 2008

[11] Jerome B. Daniel, “MECHANICAL DESIGN AND CONTROL OF THE PENDUBOT”, B.S., University of Illinois, 1991

[12] National Instruments, Introducción a la tecnología fpga: Los cinco beneficios principales, http://zone.ni.com/devzone/cda/tut/p/id/8259, 2011.

[13] López V. M.L., Ayala R. J., “FPGA: Nociones básicas e implementación”, Universidad Politécnica de Madrid, 2004

[14] National Instruments, “Introducción a la plataforma de LABVIEW Embedded”, http://zone.ni.com/wv/app/doc/p/id/wv-598, 2008

[15] National Instruments, “How to program FPGAs without any VHDL knowledge”

[16] Datasheet CD4050BC

[17] Datasheet DAC

[18] Debounce_Switch.VI, Ed Doering, ECE Department Rose-Hulman Institute of Technology, doering@rose-hulman.edu, Created 26 Aug 2009

[19] Detect_Knob_Rotation.VI, Ed Doering, ECE Department Rose-Hulman Institute of Technology, ed.doering@rose-hulman.edu, Created 12 Sep 2009

[20] Douglas L. Perry, “VHDL Programming by Example”, Fourth Edition, Mc Graw-Hill, 2002

[21] Pardo Fernando, Boluda José A. “VHDL Lenguaje para síntesis y modelado de circuitos”, Madrid España

[22] Integrating a picoblaze processor in LabVIEW FPGA by use of CLIP node, Vincent Claes, 2009

[23] LABVIEW FPGA and GPU, Universidad de Oslo, Spring 2011-Lectura11.

[24] FPGA User Manual LabVIEW, National Instruments, Abril 2003

ANEXO A

MANUAL DE USUARIO

MANUAL DE USUARIO

En el FPGA se desarrolló una interfaz en la que se puede ejecutar los modos Manual y Automático que cuenta con controles e indicadores para la ejecución del control tope y control medio y una interfaz desde el computador en el que se puede realizar también ambos tipos de control, además de cambiar el punto de operación y el valor de la matriz de ganancias del controlador.

Para poner en funcionamiento al sistema, se debe realizar las conexiones adecuadas.

A.1 CONEXIÓN DE LOS COMPONENTES

La tarjeta Spartan-3E constituye el elemento central al cual se conectan el resto de componentes del sistema; y, el módulo de potencia en donde se encuentran la alimentación del Sistema.

En el Módulo de potencia tiene las conexiones de los terminales de alimentación para el motor y los encoders, además del terminal del bus de datos para el FPGA, la forma de conexión se muestra en la figura A.1

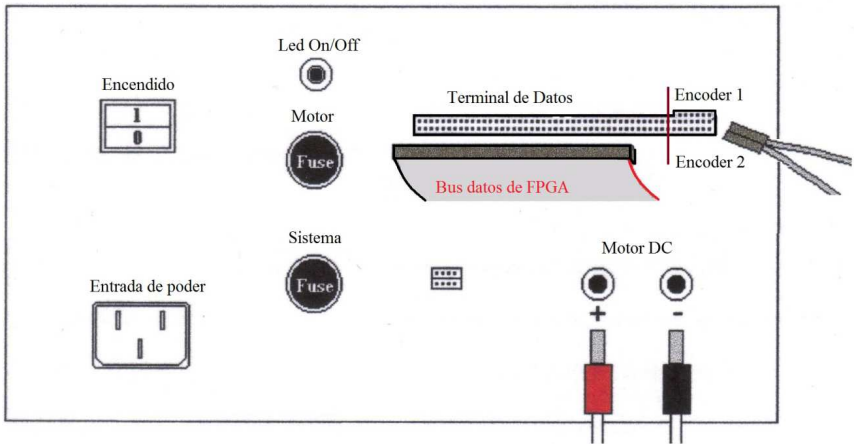


Figura A.1 Conexión de Componentes en Módulo de Potencia

Para la conexión de los encoders ópticos se debe observar que el conector del encoder 1 quede ubicado en la parte superior del terminal y el encoder 2 en la parte inferior.

La conexión del motor DC se siguen los colores de alimentación: cable rojo positivo, cable negro negativo.

El bus de datos, se debe conectar como se indica en el terminal.

Para la conexión a la tarjeta Spartan-3E, se muestra los terminales para las conexiones, evitándose de esta manera su conexión inadecuada y evitar daños en la tarjeta.

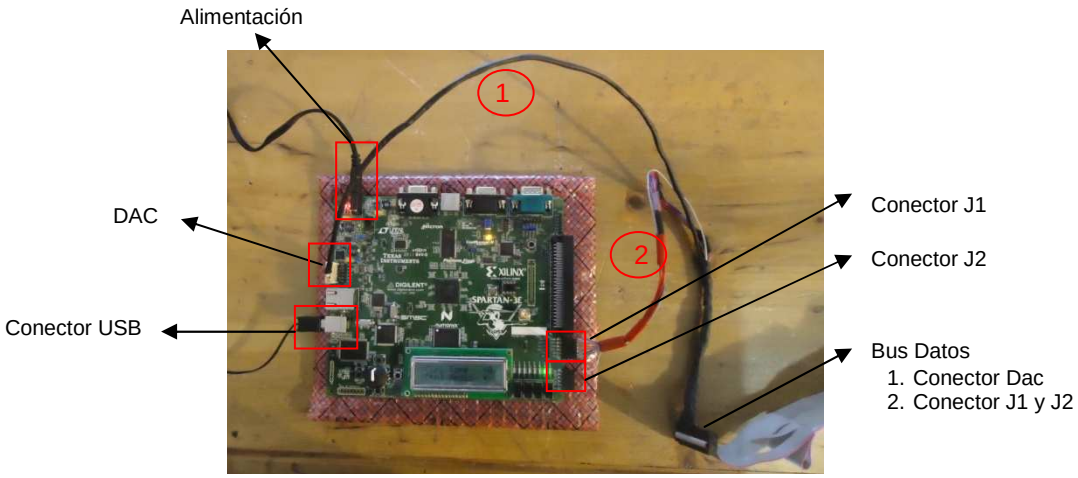


Figura A.2 Elementos de conexión con la Tarjeta Spartan-3E

La conexión de los conectores 1 y 2 se muestra en la siguiente figura:

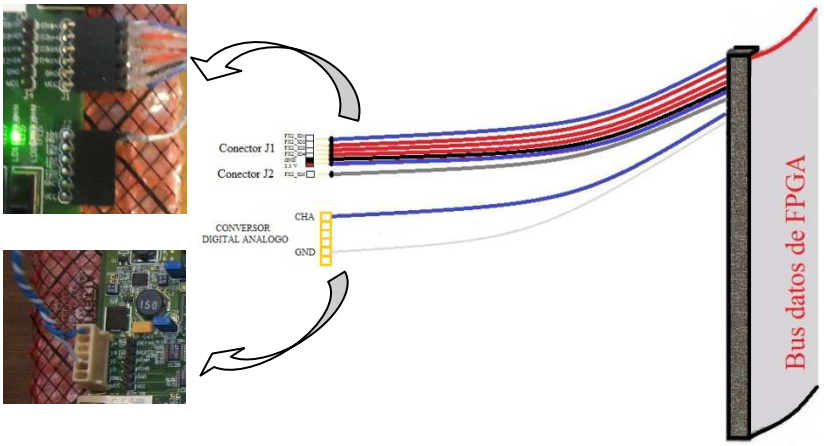


Figura A.3 Bus de datos de FPGA

Para realizar la comunicación con el computador se conecta al puerto USB de la tarjeta.

Figura A.4 Interfaz USB

A.2 DESCARGAR EL PROGRAMA EN LA TARJETA SPARTAN-3E

Si la tarjeta utilizada no contiene el programa para el control del PENDUBOT, se debe descargar el programa en la tarjeta Spartan-3E por medio del LABVIEW FPGA como se indica en la figura A.5.

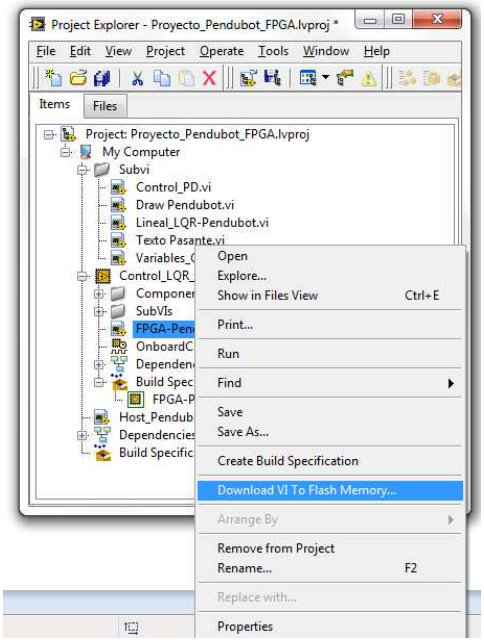


Figura A.5 Descarga de programa a Spartan-3E

Una vez revisada las conexiones y listo el programa se realiza la ejecución sea por medio de la interfaz en la FPGA o en el Computador.

A.3 EJECUCIÓN DEL PROGRAMA

El programa se inicia al momento de encender el FPGA, deben tomarse en cuenta las siguientes consideraciones antes de ejecutar las opciones de control, que se indican en la interfaz de la FPGA (por medio del LCD) o en la interfaz con el computador.

- El Pendubot debe ser ubicado en la posición estable de equilibrio $\left(-\frac{\pi}{2}, 0\right)$, se puede utilizar el nivel ubicado en el eslabón 1 para realizar esta acción.
- Se debe Encerar las posiciones, presionando el botón BTN_NORTH.

Luego de haber realizado las acciones anteriores, se debe poner en línea al sistema, cambiando la posición del SW0 en donde se indicará con el LED on/off.

A.3.1 INTERFAZ FPGA

Los pulsadores y la perilla ayudarán con el tipo de control, encerar los eslabones en las posiciones iniciales, detener el control y variar el set-point.

- BTN_NORTH: Encerar encoders
- BTN_SOUTH: Detener el control
- BTN_EAST: Control Tope
- BTN_WEST: Control Medio
- ROT_CENTER: Encerar el Set-point.
- Perilla (ROT_A y ROT_B): Variación del punto de operación (Set-point)

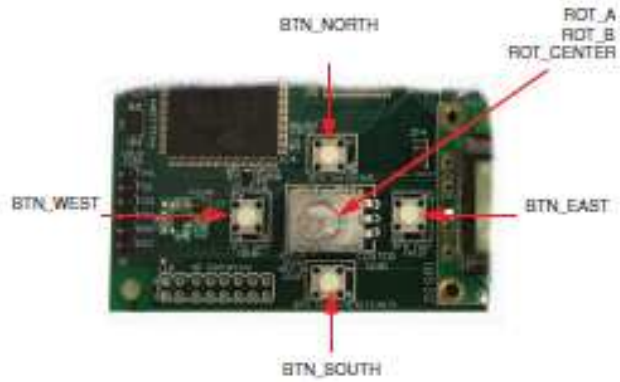


Figura A.6 Puladores y Perilla [9]

Los Switch ayudan a iniciar el proceso de control, permitiendo el control Manual/Automático y mostrar en el LCD el controlador utilizado y la posición de los eslabones.

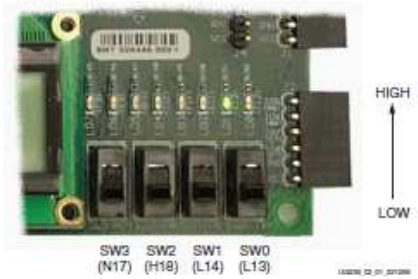


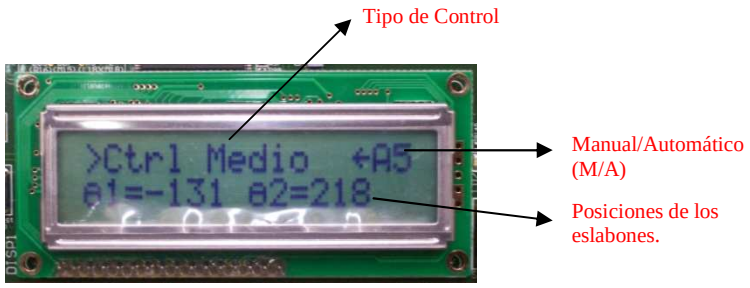
Figura A.7 Switch [9]

- SW0: Encender Control.
- SW1: Control Manual/Automático.
- SW2: Mostrar posiciones de los eslabones.

- SW3: Mostrar Ganancias del Control LQR.

Los Leds sirven como indicadores y el LCD permitá observar el tipo de control, las posiciones de los eslabones y la matriz de ganancia *K*.

- LED0: ON/OFF
- LED1: Movimiento Encoder 1
- LED2: Movimiento Encoder 2
- LED3: Control en Ejecución
- LED4: Control Tope
- LED5: Control Medio
- LED6: Manual/Automatico
- LED7: FPGA ON



(a)

(b)

Figura A.8 (a) LCD, (b) Leds indicadores

El sistema de control permite el control del sistema Pendubot en modo manual o automático.

A.3.1 Modo Automático

Esta opción permite operar al sistema automáticamente a la posición media o a la posición tope, sin necesidad de intervenir en el balanceo del sistema. Para ello el SW1 debe estar en bajo o en la posición por default del switch.

A.3.2 Modo Manual.

Esta opción permite operar al sistema manualmente, es decir que no funciona el control de balanceo y por ello se debe llevar manualmente a los eslabones alrededor del punto de operación y luego presionar el tipo de control ya sea este tope o medio. Para ello el SW1 debe estar en alto.

A.3.1 INTERFAZ COMPUTADOR (HOST)

Para la ejecución de la interfaz en el computador, se requiere tener instalado LABIEW v10.0 y los ToolKits de Real-Time y FPGA.

El archivo a ejecutarse esta en el proyecto: Pendubot_FPGA y el archivo a ejecutarse el HOST_Pendubot.

>>Project: Proyecto_Pendubot_FPGA\HOST_Pendubot

La pantalla principal se muestra en la figura A.9

Figura A.9 Pantalla principal

La interfaz de usuario del computador tiene los siguientes controles:

1. Manual/Automático.
2. Control Medio.
3. Control Tope.

4. Detener el Control
5. Continuar/Regresar Pantalla.
6. Variación del Punto de Operación (Set-point)
7. Modelo Linealizado
8. Control PD
9. Encerar o reset Encoders.
10. Reset matriz de ganancia *K*
11. Led Indicador de sistema en Línea
12. Stop Interfaz.
13. Manual de usuario

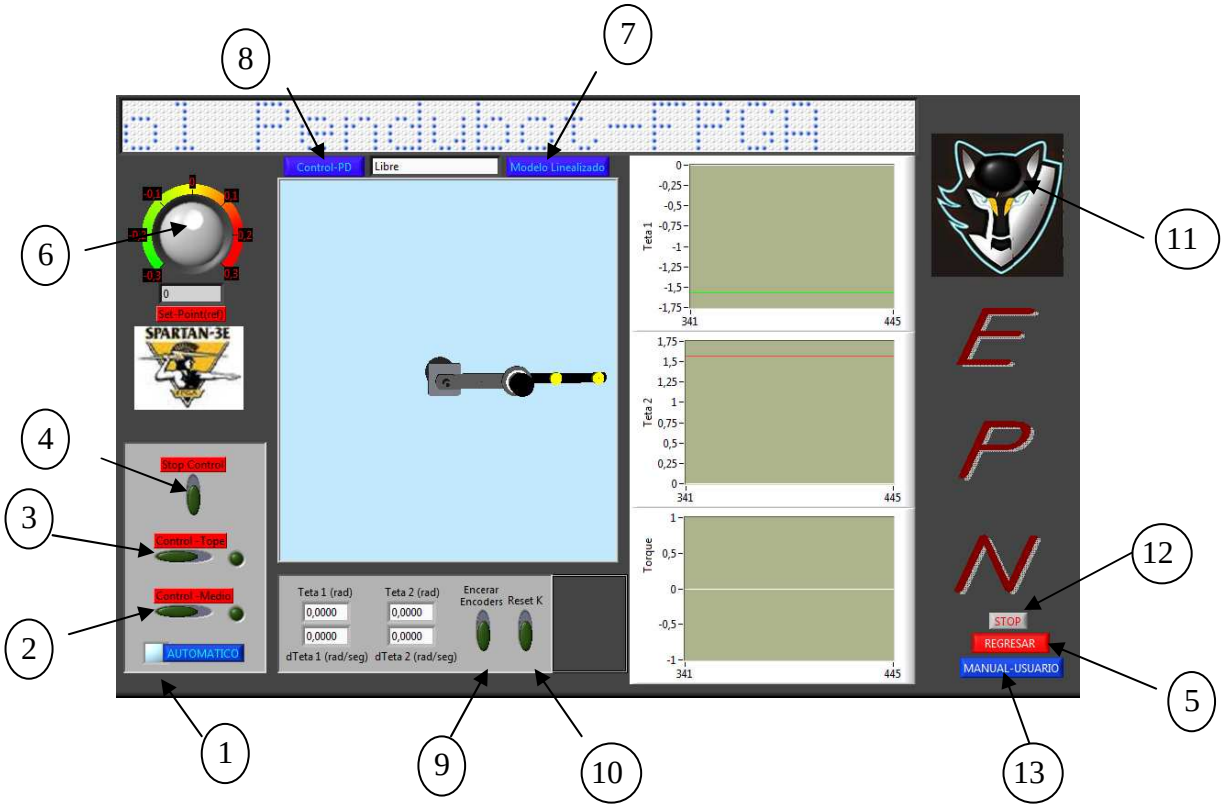


Figura A.10 Host_FPGA y controles de interfaz

Los controles anteriores excepto los botones de Modelo Linealizado y Control PD realizan las mismas acciones que si se presionaran los Pulsadores o Switch del FPGA.

El botón Modelo Linealizado permite observar las Ecuaciones de Estado del Modelo Linealizado en un punto de operación, además permite variar las matrices de ponderación y así modificar la matriz de Ganancias *K* del control LQR del

sistema, asimismo se puede cambiar el punto de linealización del sistema y poder así tener diferentes puntos de operación.

Las matrices de ponderación Q y R se determinan de la siguiente manera:

- Los elementos de la matriz Q se seleccionan para ponderar la importancia relativa de los diferentes componentes del vector de estado. La variación es en la diagonal de la matriz.

- La matriz R se selecciona para ponderar el gasto de energía en la acción de control, es decir, mientras mayor sea la ponderación asignada, la magnitud de la señal de control será menor.

La interfaz del Modelo Linealizado tiene los siguientes controles e indicadores:

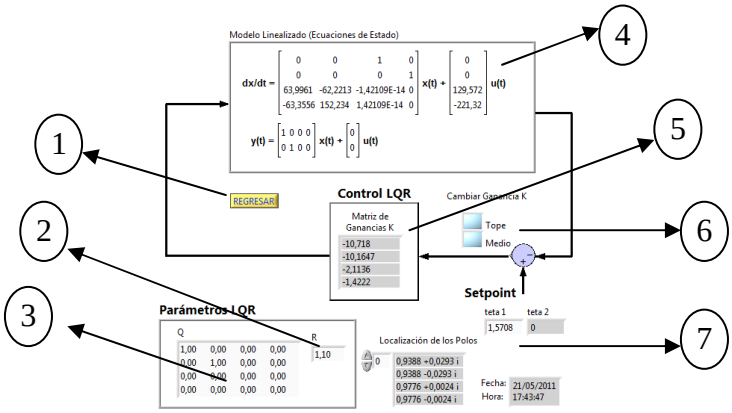


Figura A.11 Modelo Linealizado

4. Botón de regreso a la pantalla de Control.
5. Matriz R.
6. Matriz Q.
7. Visualización del Modelo Linealizado.
8. Matriz de Ganancia K nuevo.
9. Botón para cambiar a la nueva matriz K.
10. Punto de linealización del Sistema.

El botón Control PD permite observar el controlador PD utilizado para el balanceo del sistema, además de poder ajustar las constantes *Kp* y *Kd*.

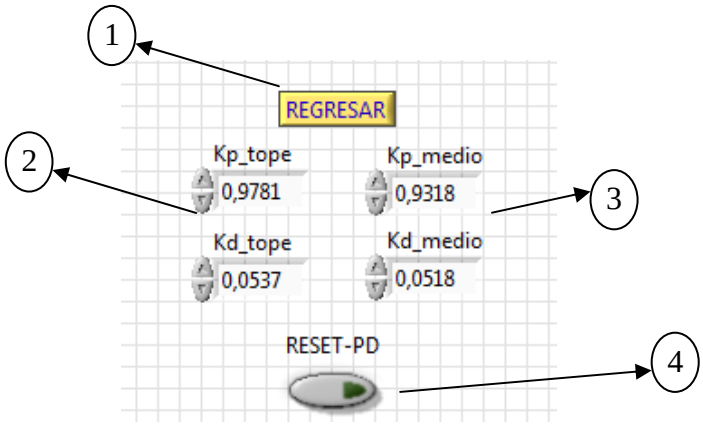


Figura A.12 Control PD

La interfaz del Control PD tiene los siguientes controles:

1. Botón de regreso a la pantalla de Control.
2. PD de Balanceo Tope.
3. PD de Balanceo Medio.
4. Regresar a valores por defecto.

Las ganancias del control PD se ajustan de la siguiente manera:

- Si el sistema no logra llegar alrededor del punto de operación, se debe disminuir Kd o aumentar Kp.
- Si el sistema se pasa alrededor del punto de operación, se debe aumentar Kd o dismiuir Kp.

La variación de las ganancias se realiza en el orden de las centésimas, es decir pequeñas variaciones.

ANEXO B

PROGRAMACIÓN BÁSICA-FPGA VI

PROGRAMACIÓN CON LABVIEW FPGA

El Módulo de LabVIEW FPGA de National Instruments extiende las capacidades de desarrollo gráfico de LabVIEW. Con el Módulo de LabVIEW FPGA, se puede crear sistemas de medición y control personalizados en hardware.

Requerimientos para trabajar con el Módulo LabVIEW FPGA y la tarjetaSpartan-3E.

- Módulo de LABVIEW FPGA.
- Módulo LabVIEW Real-Time.
- Driver de la Tarjeta Spartan-3E.

B.1 CREACIÓN DE UN FPGA-VI

Para crear un proyecto con la Spartan-3E se siguen los siguientes pasos:

1. Se abre un proyecto nuevo *Empty Project.*
2. Se da click-derecho sobre *My Computer* y en este se selecciona el dispositivo: *New» Targets and Devices,* como se muestra en la figura B.1

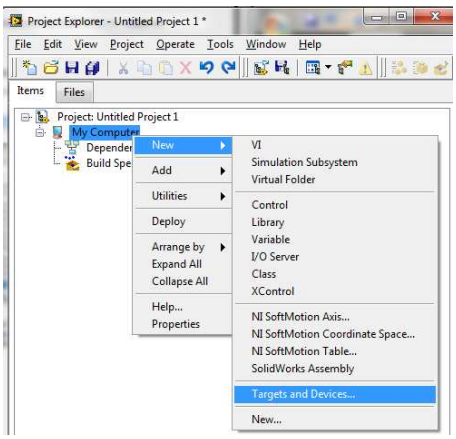


Figura B.1 Creación de un proyecto Spartan-3E

3. Luego se abre la ventana *Add Targets and Devices* y se selecciona *New target or device*, se expande *Xilinx University Program* y se selecciona la tarjeta *Spartan-3E Starter Board* y se presiona *OK*

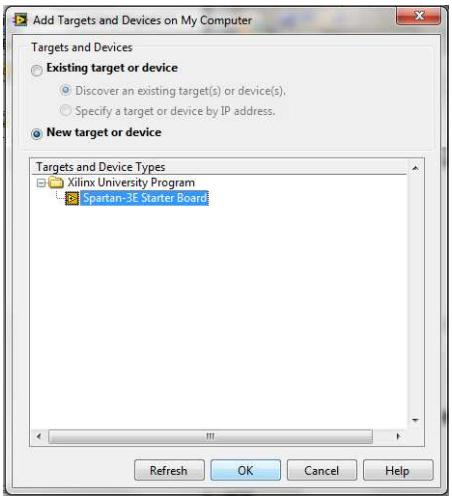


Figura B.2 Add Spartan-3E Target

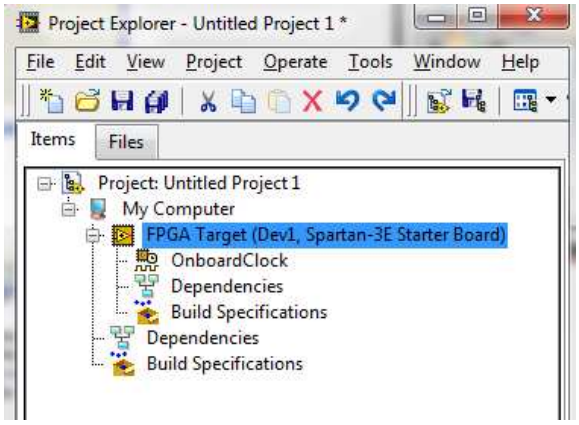


Figura B.3 Dispositivo agregado al proyecto

4. Ahora que se tiene agregado el dispositivo en el proyecto figura B.3, se crea una FPGA VI, se da click-derecho sobre el nombre del dispositivo agregado *FPGA Target (Dev1, Spartan-3E Starter Board)* y se selecciona *New» VI* donde se tendrá FPGA VI añadido al proyecto, se debe notar que el FPGA VI esta en el árbol del dispositivo FPGA.

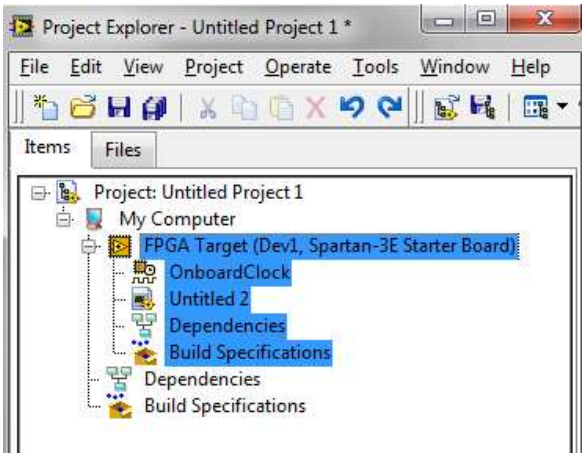


Figura B.4 FPGA VI añadido al proyecto

Una vez creado el proyecto, se deben agregar las Entradas y/o salidas que se necesiten, para esto se realizan los siguientes pasos:

1. Se da click-derecho en el nombre del dispositivo agregado *FPGA Target (Dev1, Spartan-3E Starter Board)* y se selecciona *New» FPGA I/O* como se muestra en la figura B.5
2. Ahora se observa una nueva ventana, en la cual se encuentran todos las entradas y salidas del dispositivo, además de las variables para configurar los módulos existentes en la tarjeta como son el conversor digital-análogo, conversor análogo-digital, LCD, etc.
3. Se selecciona el elemento a agregar y se lo añade en la tabla, una vez seleccionado todos los elementos que se utilicen, se presiona *OK* y se tiene como se muestra en la figura B.6

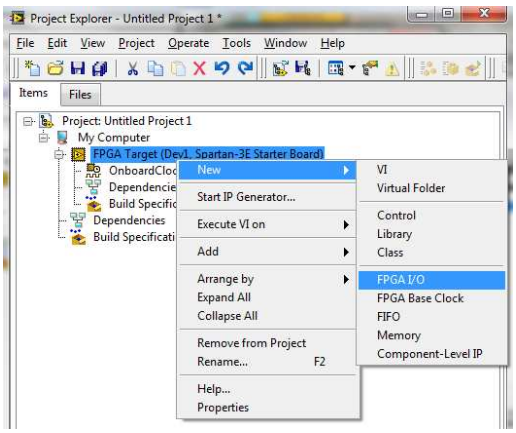


Figura B.5 Agregar E/S

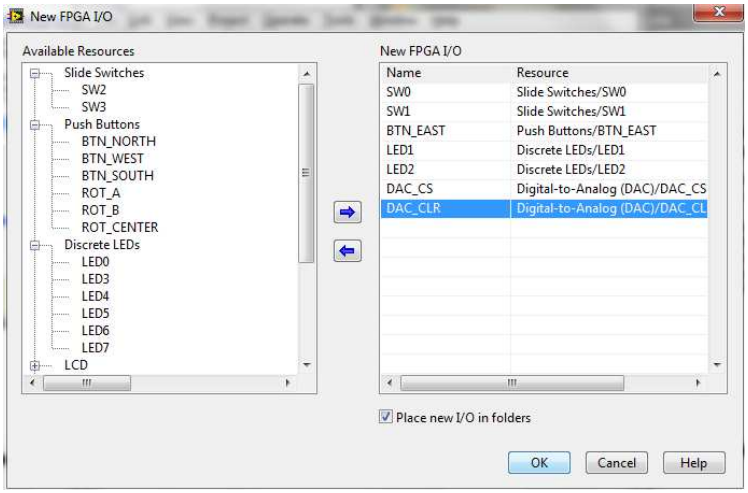


Figura B.6 Selección de E/S

Estas entradas y salidas están agregadas al proyecto, ahora se las agrega al diagrama de bloques para su utilización:

1. Como ya se tiene el FPGA VI, se abre el diagrama de bloques, se arrastra y suelta la E/S que se desee utilizar desde el *Project Explorer* hacia el diagrama de bloques. Figura B.7

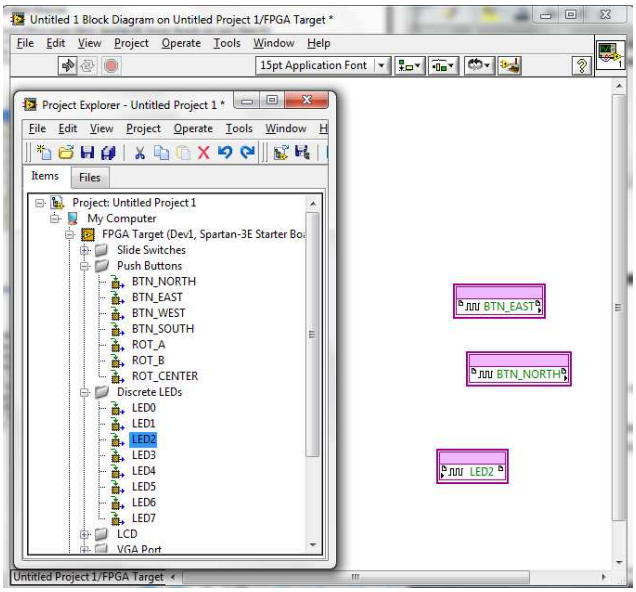


Figura B.7 E/S agregadas al Diagrama de Bloques

B.2 PALETA DE FUNCIONES LABVIEW FPGA

Cuando se selecciona un proyecto con FPGA, LABVIEW muestra solo las opciones disponibles para una FPGA específica, esto es, que solo muestra las funciones y subpaletas que se pueden utilizar con ese dispositivo, como se muestra en la figura B.8

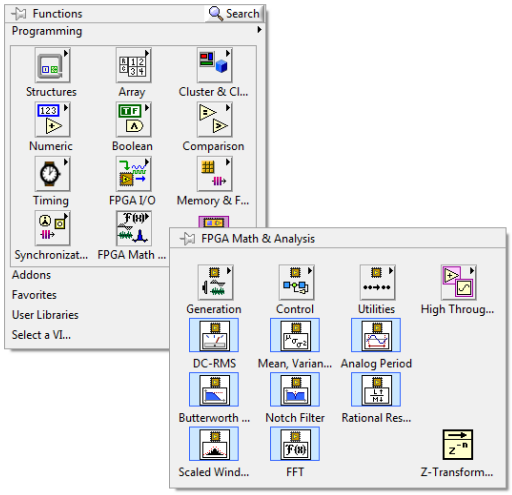


Figura B.8 Paleta de Funciones FPGA

Las *funciones* que incluye LABVIEW FPGA son: operaciones booleanas, comparaciones y operaciones matemáticas básicas.

Esta paleta ofrece todas las posibilidades de funciones que se pueden utilizar en el diagrama de bloques dentro del entorno de LabVIEW FPGA, donde al hacer click se escoje y ubica dentro del programa.

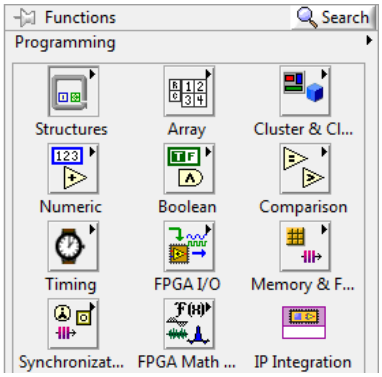


Figura B.9 Paleta de Funciones [24]

Las Funciones contenidas en esta Paleta Functions>>Programming son:

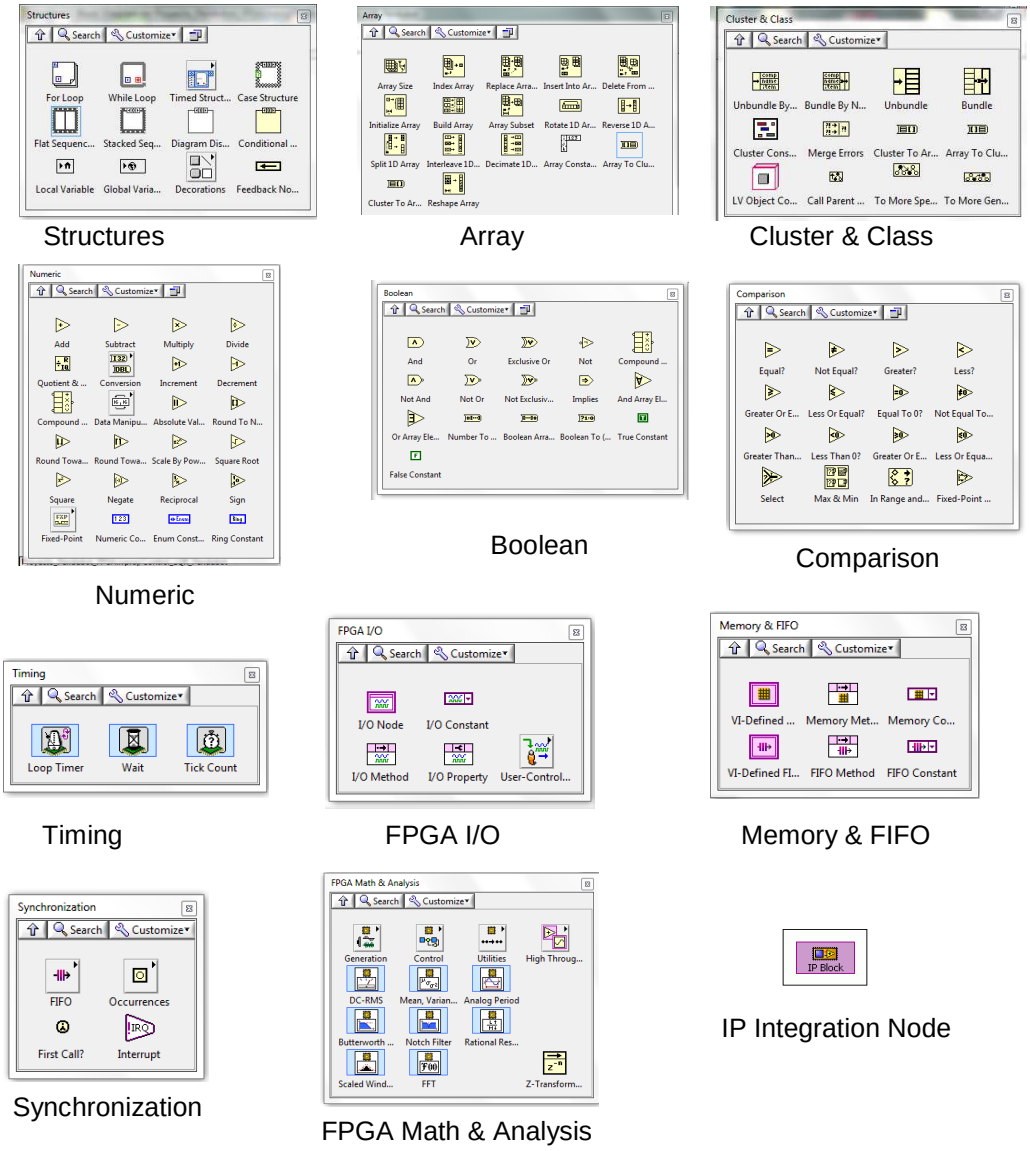


Figura B.10 Funciones de la paleta Functions de LabVIEW-FPGA

- ***Structures,*** para el control del flujo de datos.
- ***Array,*** para crear y manipular conjunto de datos del mismo tipo y de tamaño fijo.
- ***Cluster & Class*** para crear y manipular conjunto de datos de diferente tipo y de tamaño fijo.
- ***Numeric,*** para realizar operaciones aritméticas de tipo entero con signo y sin signo.
- ***Boolean,*** para realizar operaciones lógicas.

- ***Comparison,*** para comparar valores booleanos, aritméticos, arrays y clusters.
- ***Timing,*** para controlar el tiempo de ejecución de operaciones del FPGA.
- ***FPGA I/O,*** para realizar operaciones de lectura/escritura y configuraciones de los puertos de la tarjeta FPGA
- ***Memory & FIFO,*** para acceder a la memoria del FPGA y la función FIFO (first input-first output) para transferir datos.
- ***Synchronization,*** para sincronizar tareas de ejecución en paralelo y pasar datos entre tareas en paralelo.
- ***FPGA Math & Analysis,*** para realizar operaciones matemáticas de alto rendimiento (HighThroughput Math) y operaciones de control sobre el FPGA.

  Las operaciones de alto rendimiento permiten realizar operaciones de punto flotante.

  Las operaciones de Control permiten realizar análisis de señales, generar señales de onda cuadrada, senoidales y cosenoidales y crear aplicaciones de control (PID).
- ***Ip Integration Node,*** integra código IP (intellectual property), es decir se maneja código en VHDL dentro de LabVIEW como módulos adicionales. Antes de que se integre este código, es necesario que este previamente compilado.

B.3 CREACIÓN DE HOST-VI

El Host se debe crear dentro del mismo proyecto del FPGA. Se siguen los siguientes pasos:

1. Se hace click-derecho sobre *My Computer* en la ventana del proyecto y se selecciona *New»VI.* Se nota que el nuevo VI se encuentra bajo el árbol de *My Computer.*

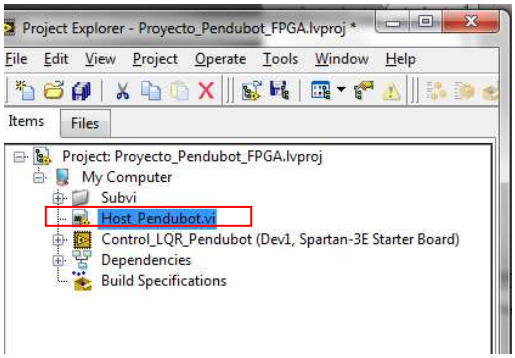
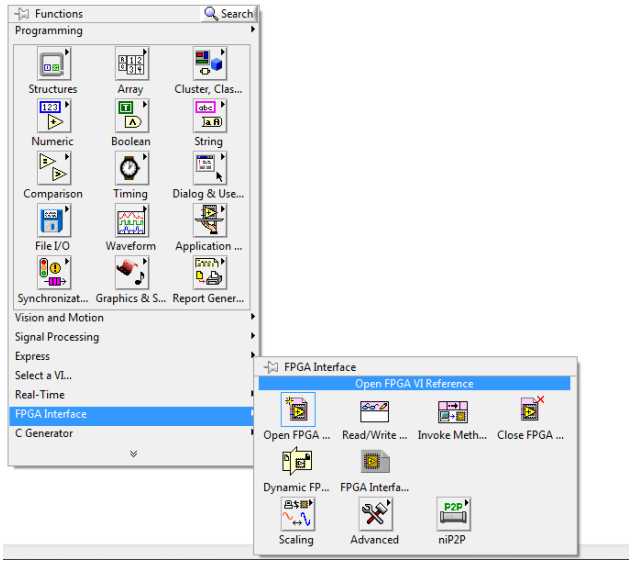


Figura B.11 HOST_FPGA

2. Sobre el diagrama de bloques del nuevo VI, se coloca la función *>>Open FPGA VI Reference*, la función se encuentra en la paleta de funciones en la interfaz de FPGA. El *Open FPGA VI Reference* se utiliza para hacer referencia al FPGA-VI situado en el mismo proyecto. Sobre ésta función se hace click derecho y se selecciona el FPGA-VI, como se muestra en la Figura B.13.



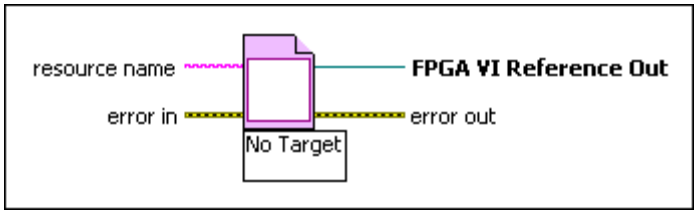


Figura B.12 HOST_FPGA, *Open FPGA VI Reference*

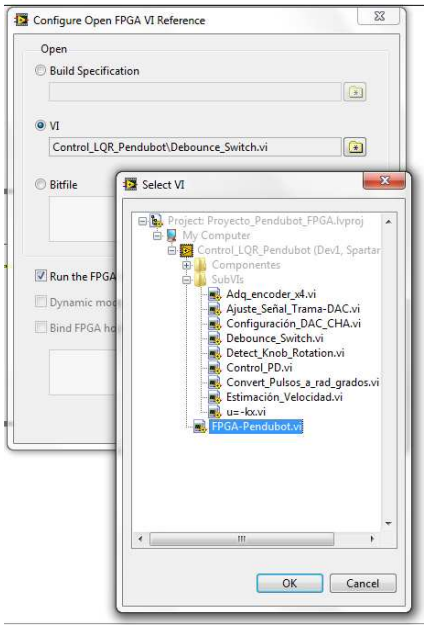


Figura B.13 Selección del FPGA-VI

3. Se coloca la función *Read/Write (Function>>FPGA Interface>>Read/Write)* el cual hace referencia a los terminales de control del FPGA-VI creado.

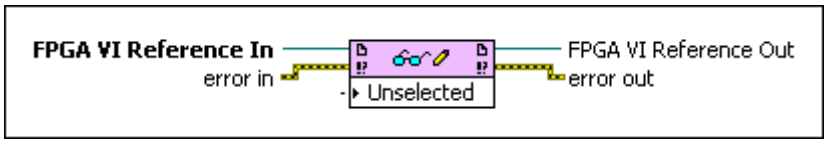


Figura B.14 Read/Write

4. Se hace click sobre la función *Read/Write* y se selecciona los controles que se deseen observar y/o manipular en el Host VI.
5. Se cierra el recurso abierto en la función función *Function>>FPGA Interface>>Close FPGA VI Reference.*

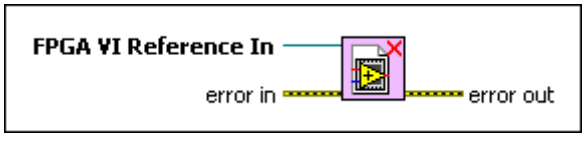


Figura B.15 Close FPGA VI Reference

Para que el HOST-FPGA no aborte la ejecución sobre el FPGA cuando se desconecte el host o se termine la ejecución del mismo, se configura esta función, por ello se selecciona *Close.*

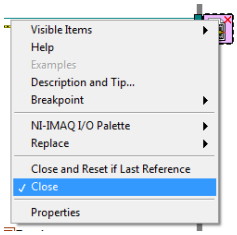


Figura B.16 Configuración función *Close FPGA VI Reference*

6. El Host VI debe ser similar a la Figura B.17.

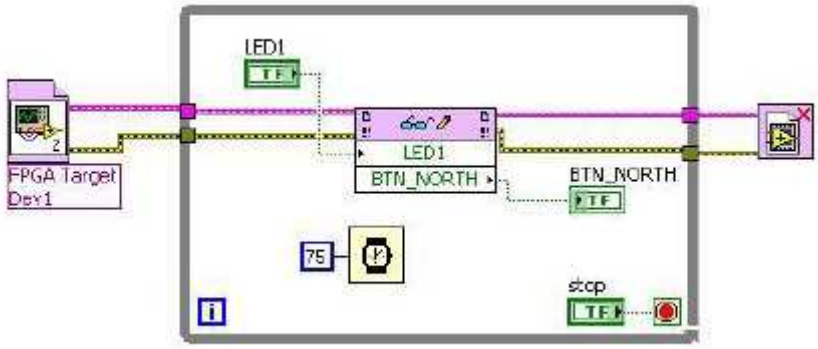


Figura B.17 Host VI

Este VI permite manipular los controles del FPGA VI en la tarjeta de desarrollo.

B.4 COMPILACIÓN FPGA-VI

Para poder descargar el sistema de control al FPGA, es necesario compilar el VI. Al compilar el archivo, LABVIEW FPGA determina el área utilizada en el FPGA, según el número de slices, flip-flops, lookup tables (LUTs), multiplicadores y bloques de memoria RAM utilizados en el diseño, para así conocer el tamaño de implementación en la tarjeta.

Antes de compilar el archivo, es necesario configurar el modo de la FPGA, es decir se debe especificar que el FPGA VI corra o arranque automáticamente cuando se descargue sobre el dispositivo, ya que al no hacerlo no se tendrá independencia con la PC.

La configuración se muestra en la figura B.18

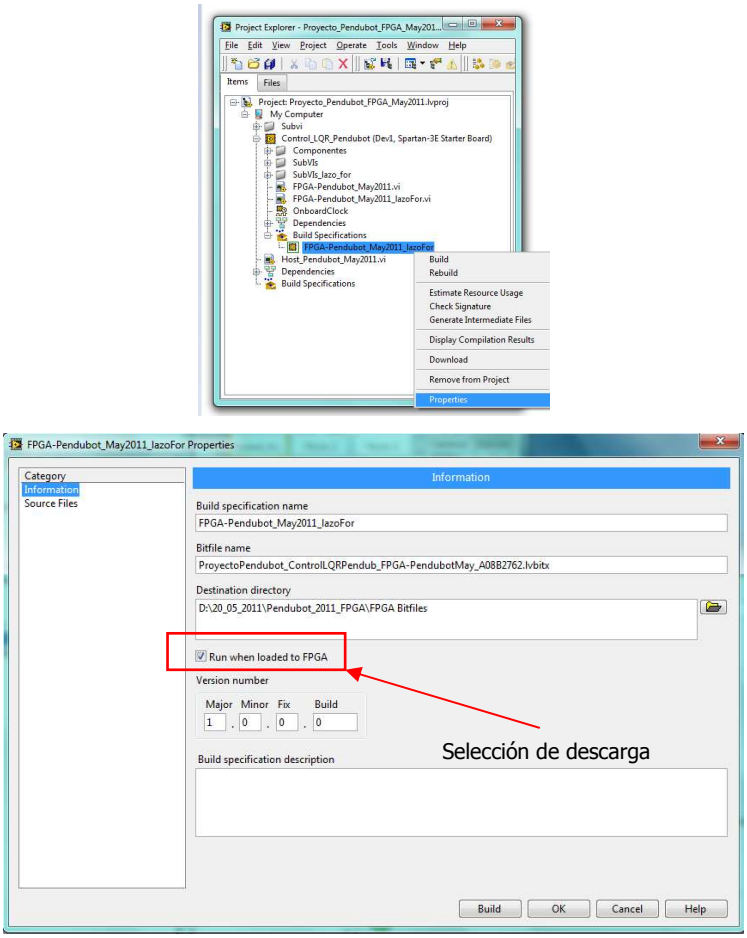


Figura B.18 Propiedades de configuración del FPGA VI

Finalizada la compilación, se crean los archivos necesarios para descargar a la tarjeta, El archivo que se descarga a la tarjeta tiene extensión *.lvbitx* y se crea en la carpeta *FPGA Bitfiles* de la carpeta raíz donde se almacena el proyecto principal.

El estado de compilación se muestra en el cuadro de diálogo y cuando se ha visualizado el mensaje de la figura B.19 "*The compilation complete successfully*", el archivo estará generado.

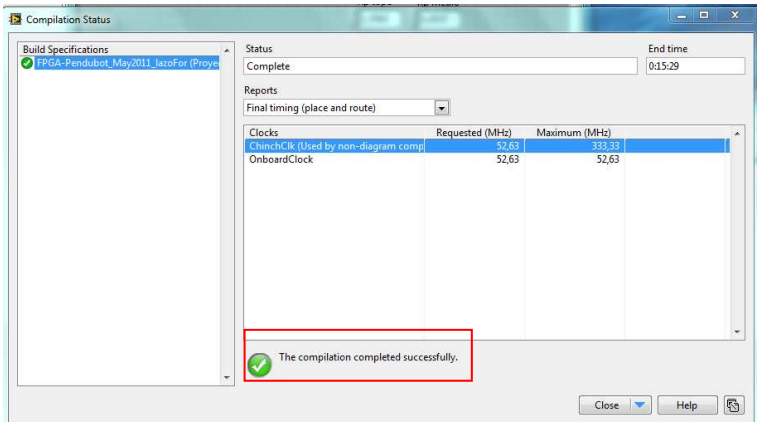


Figura B.19 Compilación terminada.

B.5 DESCARGA FPGA-VI

LABVIEW FPGA permite descargar directamente el archivo a la tarjeta luego de haber terminado la compilación, cabe indicar que se debe descargar el archivo a la memoria Flash de la tarjeta, ya que este proyecto implica que el control se debe realizar sin utilizar la interfaz con la PC, es decir que el programa de control se ejecuta directamente en la FPGA de manera autónoma.

Para ello se selecciona el FPGA VI que se desee descargar, como se muestra en la figura B.20

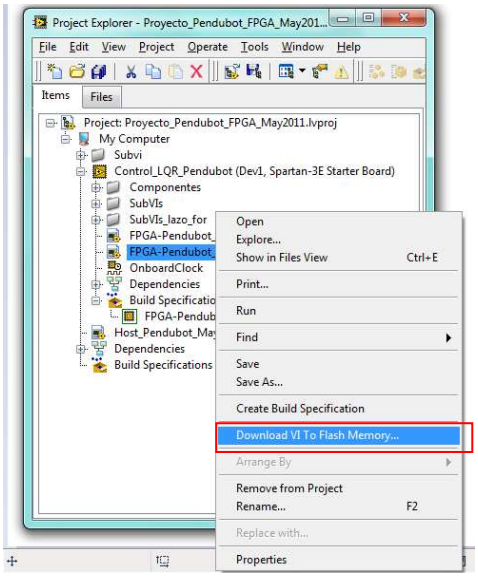


Figura B.20 Descarga.

El mensaje que asegura la descarga del archivo es “*Download Successful*” como se indica en la figura B.21

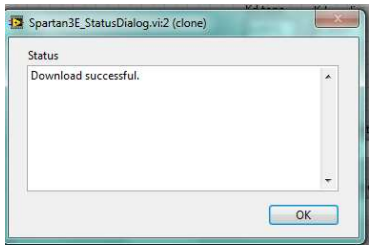


Figura B.21 Descarga exitosa del archivo .lvbitx a la tarjeta Spartan-3E

ANEXO C

ARQUITECTURA FPGA

ARQUITECTURA DE LA FPGA SPARTAN 3 DE XILINX

Las FPGA Spartan III de Xilinx están conformadas por un conjunto de Bloques Lógicos Configurables (Configurable Logic Blocks: CLBs) rodeados por un perímetro de Bloques Programables de entrada/salida (Programmable Input/Output Blocks: IOBs). Estos elementos funcionales están interconectados por una jerarquía de canales de conexión (Routing Channels), la que incluye una red de baja capacitancia para la distribución de señales de reloj de alta frecuencia. Adicionalmente el dispositivo cuenta con 24 bloques de memoria RAM de 2Kbytes de doble puerto, cuyos anchos de buses son configurables, y con 12 bloques de multiplicadores dedicados de 18 X 18 bits.

Los cinco elementos funcionales programables que la componen son los siguientes:

Bloques de entrada/salida (Input/Output Blocks – IOBs): Controlan el flujo de datos entre los pines de entrada/salida y la lógica interna del dispositivo. Soportan flujo bidireccional más operación tri-estado y un conjunto de estándares de voltaje e impedancia controlados de manera digital.

Bloques Lógicos configurables (Configurable Logic Blocks – CLBs): Contienen Look-Up Tables basadas en tecnología RAM (LUTs) para implementar funciones lógicas y elementos de almacenamiento que pueden ser usados como flip-flops o como latches.

Bloques de memoria RAM (Block RAM): Proveen almacenamiento de datos en bloques de 18 Kbits con dos puertos independientes cada uno.

Bloques de multiplicación que aceptan dos números binarios de 18 bit como entrada y entregan uno de 36 bits.

Administradores digitales de reloj (Digital Clock Managers – DCMs): Estos elementos proveen funciones digitales auto-calibradas, las que se encargan de distribuir, retrasar arbitrariamente en pocos grados, desfasar en 90, 180, y 270 grados, dividir y multiplicar las señales de reloj de todo el circuito.

Los elementos descritos están organizados como se muestra en la Figura . Un anillo de IOBs rodea un arreglo regular de CLBs. Atraviesa este arreglo una columna de Bloques de memoria RAM, compuesta por varios bloques de 18 Kbit, cada uno de los cuales está asociado con un multiplicador dedicado. Los DCMs están colocados en los extremos de dichas columnas.

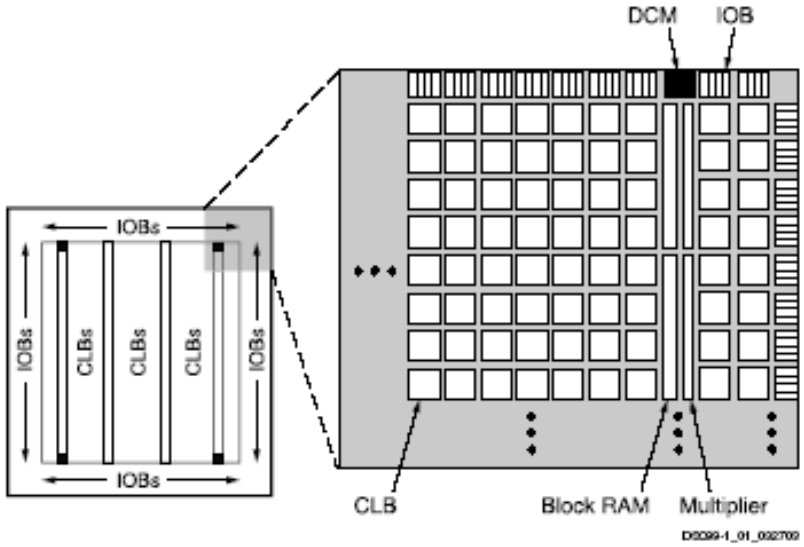


Figura 1: Arquitectura de la Spartan 3

A continuación se hace una descripción más detallada de cada uno de los elementos funcionales de la FPGA, y luego se describe el proceso de configuración de la misma.

Bloques de entrada/salida IOB

Los bloques de entrada/salida (IOB) suministran una interfaz bidireccional programable entre un pin de entrada/salida y la lógica interna de la FPGA. Un diagrama simplificado de la estructura interna de un IOB aparece en la Figura 2. Hay tres rutas para señales en un IOB: la ruta de salida, la ruta de entrada y la

ruta tri-estado. Cada ruta tiene su propio par de elementos de almacenamiento que pueden actuar tanto como registros o como latches. Las tres rutas principales son como sigue:
La ruta de entrada, que lleva datos desde el pad, que está unido al pin del package, a través de un elemento de retardo opcional programable, directamente a la línea I. Después del elemento de retardo hay rutas alternativas a través de un par de elementos de almacenamiento hacia las líneas IQ1 e IQ2. Las tres salidas del IOB todas conducen a la lógica interna de la FPGA.

La ruta de salida, que parte con las líneas O1 y O2, lleva datos desde la lógica interna de la FPGA, a través de un multiplexor y del driver tri-estado hacia el pad del IOB. En suma a esta ruta directa, el multiplexor da la opción de insertar un par de elementos de almacenamiento.

La ruta tri-estado determina cuando el driver de salida está en alta impedancia. Las líneas T1 y T2 llevan datos desde la lógica interna a través de un multiplexor hacia el driver de salida. En suma a esta ruta directa, el multiplexor da la opción de entregar un par de elementos de almacenamiento.

Todas las rutas de señales que entran al IOB, incluidas aquellas asociadas con los elementos de almacenamiento tienen una opción de inversión. Cualquier inversor colocado (en la programación) en estas rutas es automáticamente absorbido dentro del IOB.

Hay tres pares de elementos de almacenamiento en cada IOB, un par para cada uno de las tres rutas. Es posible configurar cada uno de esos elementos como un *flip-flop* D gatillado por flanco (FD) o como un *latch* sensible a nivel (LD). Estos elementos son controlados con la misma red de distribución de relojes que se utiliza para todo el sistema.

El par de elementos de almacenamiento tanto de la ruta de salida o de la del *driver* tri-estado pueden ser usados en conjunto, con un multiplexor especial para producir transmisión de doble tasa de datos (DDR). Esto se logra tomando datos

sincronizados con el flanco de subida del reloj y convirtiéndolos en bits sincronizados tanto con el flanco de subida como con el de bajada. A esta combinación de dos registros y un multiplexor se le llama *flip flop* tipo D de doble tasa de datos (FDDR).

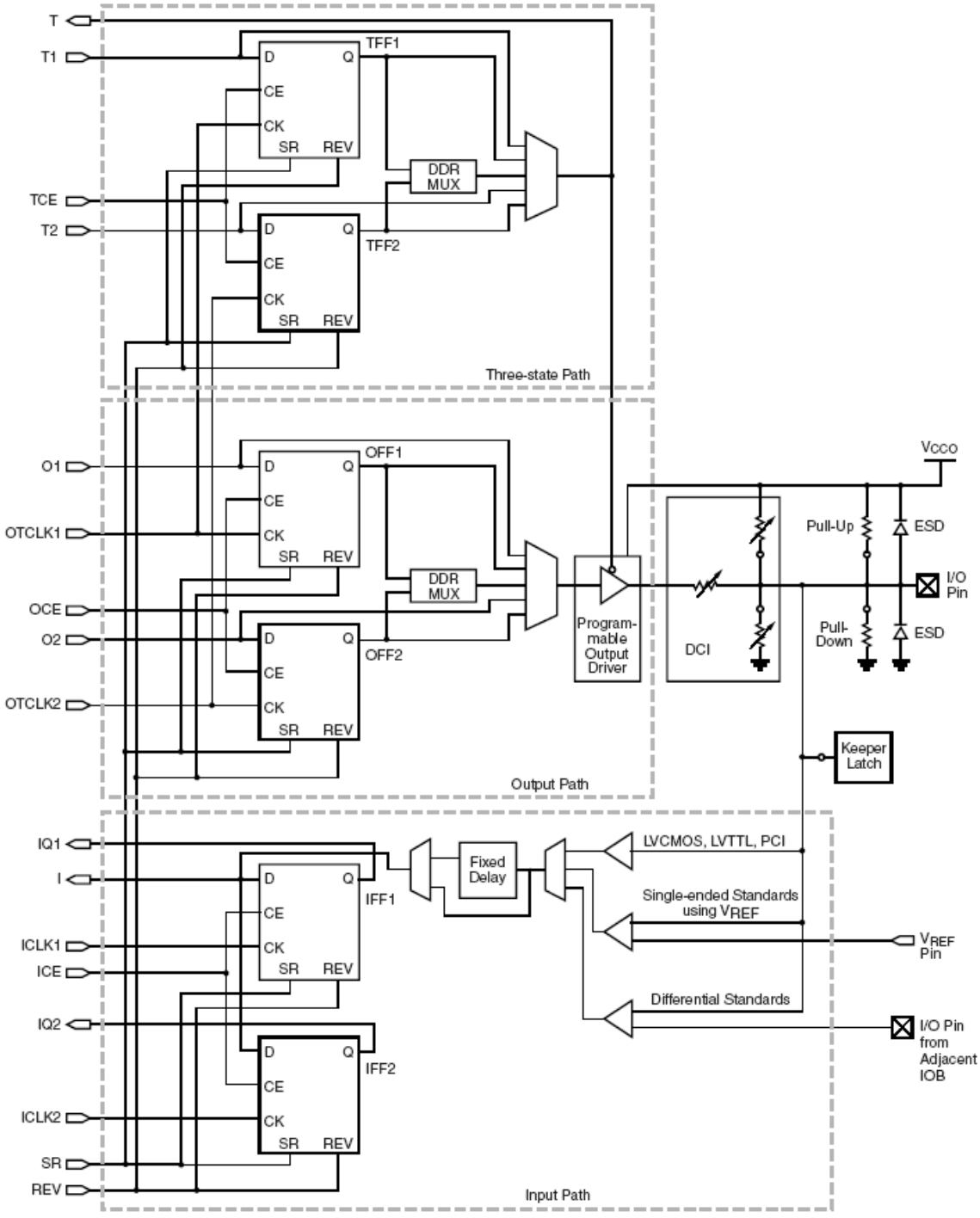


Figura 2: Diagrama simplificado de un IOB de la Spartan III

Cada IOB cuenta además con otros elementos, entre los cuales cuentan las resistencias de *Pull-Up* y de *Pull-Down*, que tienen el objetivo de establecer niveles altos o bajos respectivamente en las salidas de los IOBs que no están en uso; un circuito de retención (Keeper) del último nivel lógico que se mantiene, después de que todos los drivers han sido apagados, lo que es útil para cuidar que las líneas de un bus no floten, cuando los drivers conectados están en alta impedancia; un circuito de protección para descargas electro estáticas (protección ESD), que utiliza diodos de protección.

Finalmente cada IOB cuenta con un control para el *slew rate* y para la corriente de salida máxima. El primero otorga la posibilidad de elegir una tasa alta de cambio de nivel (con bajo *slew rate*) o una tasa máxima menor, pero con un control de transciente, para la utilización de los puertos en la integración a buses, donde al pasar de alta impedancia a un nivel de voltaje suele producirse transiciones inesperadas. El segundo entrega siete niveles deferentes de corrientes máximas tanto para el estándar CMOS como para el TTL, lo que permite adaptarse a dispositivos que necesitan mayores corrientes para su activación; en el caso del estándar LVCMOS a 2.5V el rango de corrientes es de 2 a 24 mA.(2, 4, 6, 8, 12, 16, 24 mA).

Los IOB soportan 17 estándares de señales de salida de terminación única y seis de señal diferencial; también cuentan con un sistema integrado, para coincidir con la impedancia de las líneas de transmisión que llegan a la FPGA, llamado Control Digital de Impedancia (DCI), el que permite elegir hasta 5 tipos diferentes de terminaciones, utilizando una red de resistencias internas que se ajustan en serie o en paralelo, dependiendo de las necesidades del estándar elegido.

Bloques de Lógica Configurable (CLB)

El bloque básico de la red que compone la FPGA es la *slice*. Existen dos tipo de slice, éstas se diferencian en algunos elementos, pero son muy parecidas, (ver más adelante). Luego estas *slices* se organizan en los bloques lógicos elementales, que son los que se describen a continuación.

os Bloques de Lógica Configurable (CLBs) constituyen el recurso lógico principal para implementar circuitos síncronos o combinacionales. Cada CLB está compuesta de cuatro slices interconectadas entre si, tal como se muestra en la Figura 3.

Las cuatro slices que componen un CLB tienen los siguientes elementos en común: dos generadores de funciones lógicas, dos elementos de almacenamiento, multiplexores de función amplia, lógica de *carry* y compuertas aritméticas, tal como se muestra en la Figura 4. Los dos pares de *slices* usan estos elementos para entregar funciones lógicas y aritméticas de ROM. Además de lo anterior, el par de la izquierda soporta dos funciones adicionales: almacenamiento de datos usando RAM distribuida y corrimiento de datos con registros de 16 bits.

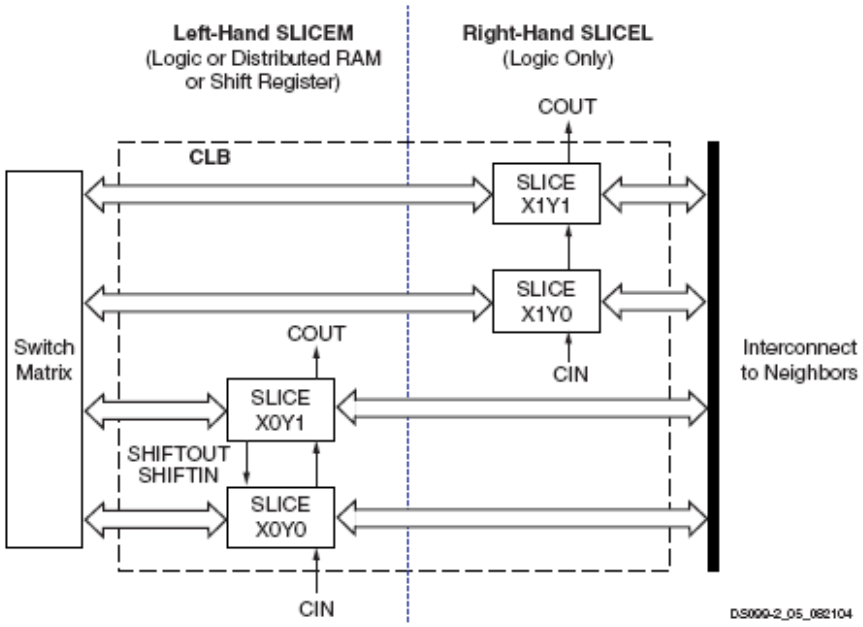


Figura 3: Arreglo de slices en un CLB

La Figura 4 es un diagrama de una slice del par del lado izquierdo, por lo tanto representa un súper conjunto de los elementos y conexiones que se encuentran el las *slices*.

El generador de funciones basado en RAM –también conocido como *Look-Up Table* (LUT)- es el recurso principal para implementar funciones lógicas dentro de

la FPGA. Más aún, las LUTs en cada par de *slices* del lado izquierdo pueden ser configuradas como RAM distribuida o como un registro de corrimiento de 16 bits. Los generadores de funciones ubicados en las porciones superiores e inferiores de la slice son referidos como “G-LUT” y “F-LUT” respectivamente en la Figura 4.

El elemento de almacenamiento, el cual es programable tanto como un *flip flop* tipo D o como un *latch* sensible a nivel, provee un medio para sincronizar datos a una señal de reloj, entre otros usos. Estos elementos de almacenamiento, que se encuentran en las porciones superiores e inferiores de la slice son llamados “FFY” y “FFX”, respectivamente.

Los multiplexores de función amplia combinan las LUTs para permitir operaciones lógicas más complejas, cada slice tiene dos de éstos, en la Figura 4 corresponden a F5MUX y F1MUX.

La cadena de *carry*, en combinación con varias compuertas lógicas dedicadas, soportan implementaciones rápidas de operaciones matemáticas. La cadena de *carry* entra a la *slice* como CIN y sale como COUT. Cinco multiplexores controlan la cadena: CYINIT, CY0F y CYMUXF en la porción inferior, así como CY0G y CYMUXG en la porción superior. La lógica aritmética dedicada incluye compuertas XOR y AND en cada porción de la slice.

Con un rol central en la operación de cada *slice* se encuentran dos rutas de datos casi idénticas. Para la descripción que prosigue se usan los nombres de la parte inferior de la Figura 4. La ruta básica tiene su origen en la matriz de *switches* de interconexión colocada fuera del CLB. Cuatro líneas, F1 a F4 entran en la slice y se conectan directamente a la LUT. Una vez dentro de la slice, la ruta de los 4 bits inferiores pasa a través de un generador de funciones F que realiza operaciones lógicas. La ruta de salida del generador de funciones, D, ofrece cinco posibles rutas posibles:

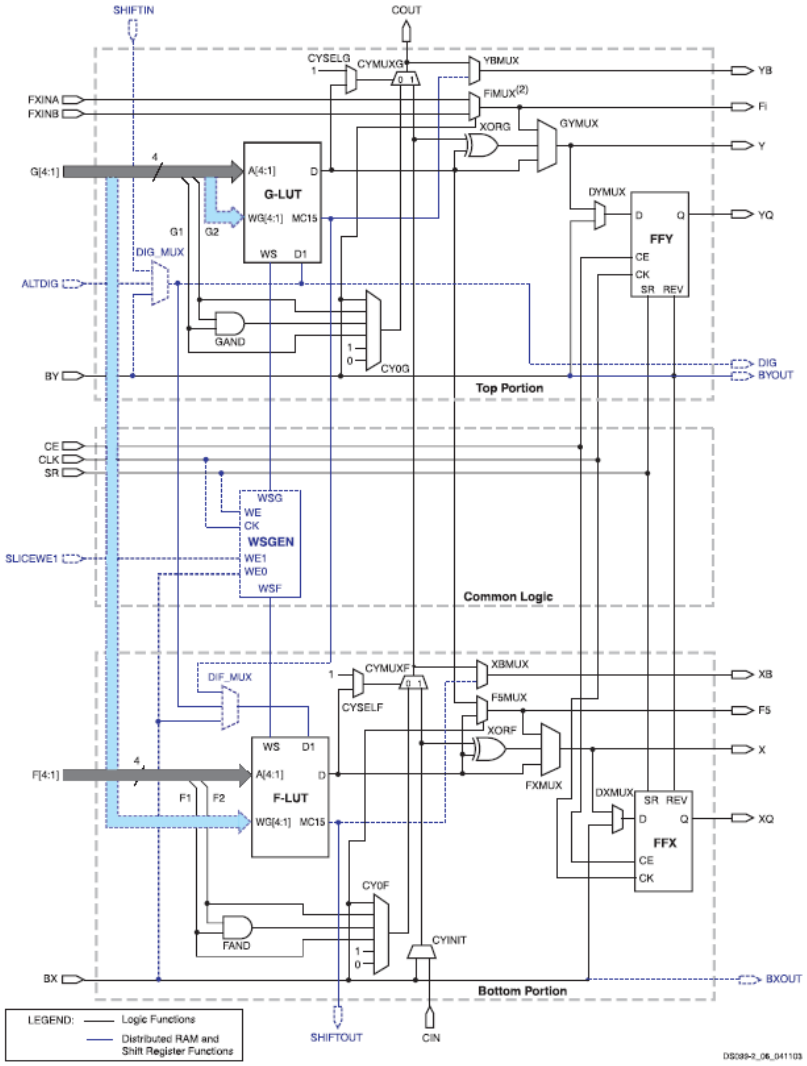


Figura 4: Diagrama simplificado de una slice del lado izquierdo de un CLB

Salir de la slice por la línea X y volver a interconectarse.

Dentro de la slice, X sirve como entrada al DXMUX que alimenta la entrada de datos D, correspondiente al elemento de almacenamiento FFY. La salida Q de este elemento maneja la ruta XQ que sale de la slice.

Controlar el multiplexor CYMUXF de la cadena de carry.

Con la cadena de carry, servir como una entrada a la compuerta XORF, que realiza operaciones aritméticas y produce el resultado en X.

Manejar el multiplexor F5MUX para implementar funciones lógicas más anchas que 4 bits. Las salidas D de los F-LUT y G-LUT sirven de entradas de datos para este multiplexor.

En suma a los caminos lógicos principales descritos recién, existen dos rutas de *bypass* que entran a la slice como BX y BY. Una vez dentro de la FPGA, BX en la parte de debajo de la slice (o BY en la parte superior) puede tomar cualquiera de varias ramas diferentes:

Hacer bypass de la LUT y del elemento de almacenamiento, luego salir de la slice como BXOUT y volver a interconectarse.
Hacer bypass a la LUT, y luego pasar a través del elemento de almacenamiento, para luego salir como XQ.
Controlar el multiplexor F5MUX.
Servir como una entrada a la cadena de carry vía los multiplexores.
Manejar la entrada DI de la LUT.
BY puede controlar la entrada REV de FFY y de FFX.
Finalmente, el multiplexor DIG_MUX puede derivar la ruta BY hacia la línea DIG que sale de la slice.

Cada una de las dos LUTs (F y G) de una *slice* tiene cuatro entradas lógicas (A1–A4) y una única salida D. Esto permite programar cualquier operación lógica booleana de cuatro variables en este dispositivo. Además, los multiplexores de función amplia pueden usarse para combinar LUTs dentro del mismo CLB o incluso a través de diferentes CLBs, haciendo posible funciones con mayor número de variables.

Las LUT de ambos pares de *slices* dentro de un CLB no sólo soportan las funciones descritas, si no que también pueden funcionar como ROM (Read Only Memory) con datos inicializados al momento de configurar la FPGA. Las LUTs del lado izquierdo de cada CLB soportan además dos funciones adicionales: primero, es posible programarlas como RAM distribuida, lo que permite contar con espacios de memoria de 16 bits en cualquier parte de la topología de la FPGA.

Segundo, es posible programar una de estas LUTs como un registro de desplazamiento de 16 bits, con lo que se pueden producir retardos de hasta 16 bits o combinaciones de varias LUTs pueden producirlos de cualquier largo de bits.

Bloques dedicados de memoria RAM

La Spartan III tiene 24 bloques de 18 Kbits de memoria RAM. El ancho del bus de datos versus el de direcciones (relación de aspecto) de cada bloque es configurable y se puede combinar varios de éstos para formar memorias más anchas o de mayor profundidad.

Tal como se muestra en la Figura 5, los bloques de RAM tienen una estructura de doble puerto. Dos puertos idénticos llamados A y B permiten acceso independiente al mismo rango de memoria, que tiene una capacidad máxima de 18.432 bits – o 16.384 cuando no se usan las líneas de paridad. Cada puerto tiene su propio set de líneas de control, de datos y de reloj para las operaciones síncronas de lectura y escritura. Estas operaciones tienen lugar de manera totalmente independiente en cada uno de los puertos.

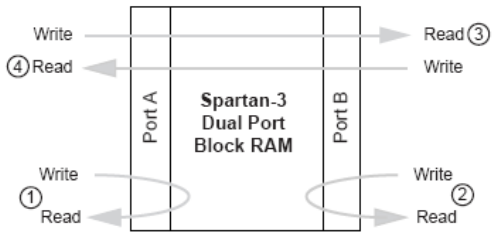


Figura 5: Diagrama de un bloque de RAM dedicado de la Spartan III

Multiplicadores dedicados

La Spartan III provee multiplicadores embebidos que aceptan palabras de 18 bits como entrada y entregan productos de 36 bits. Los buses de entrada de estos multiplicadores aceptan datos en complemento dos (tanto 18 bits con signo, como

17 bits sin signo). Para cada bloque de RAM hay un multiplicador inmediatamente colocado y conectado; dicha proximidad permite manejo eficiente de los datos.

Digital Clock Manager (DCM) y red de distribución de relojes

La Spartan III tiene 4 bloques para el control de todos los aspectos relacionados con la frecuencia, la fase y el *skew* de la red de relojes de la FPGA. Cada DCM tiene cuatro componentes funcionales: El Delay-Locked Loop (DLL), El Sintetizador Digital de Frecuencia (DFS) y el Desplazador de fase (PS). Además incluye cierta lógica para *status*.

La Figura 6 muestra un diagrama de bloques de este elemento funcional de la FPGA.

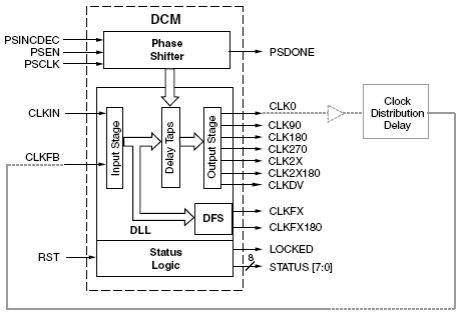


Figura 6: Diagrama de bloques de uno de los cuatro DCMs de la Spartan III

El DCM realiza tres funciones principales:

Eliminación de skew de reloj: El concepto de skew describe el grado al cual las señales de reloj pueden, bajo circunstancias normales, desviarse del alineamiento de la fase cero. Ello ocurre cuando pequeñas diferencias en los retardos de las rutas causan que la señal de reloj llegue a diferentes puntos del circuito en tiempos diferentes. Este skew de reloj puede incrementar los requerimientos de set-up time y de hold time, lo que puede perjudicar el desempeño de aplicaciones de alta frecuencia. El DCM elimina el skew de reloj alineando la salida de la señal de reloj que genera con otra versión de la misma señal que es retroalimentada. Como resultado se establece una relación de cero desfase entre ambas señales.

Síntesis de frecuencia: Provisto de una señal de reloj de entrada, el DCM puede generar diferentes relojes de salida. Ello se logra multiplicando y/o dividiendo la frecuencia del reloj de entrada.
Corrimiento de fase: El DCM puede producir desfases controlados de la señal de reloj de entrada y producir con ello relojes de salida con diferentes fases.

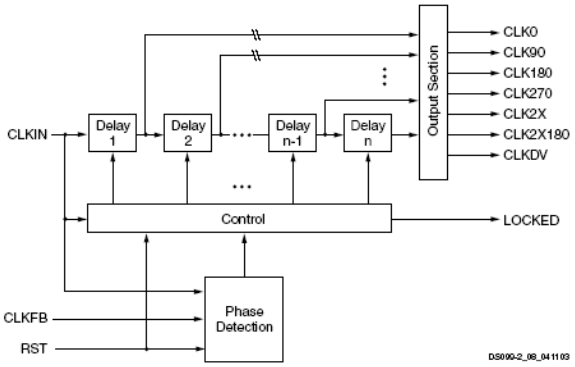


Figura 7: Diagrama funcional del Delay-Locked Loop (DLL)

El DLL tiene como principal función eliminar el *skew* de reloj. La ruta principal del DLL consiste en una etapa de entrada, seguida por una serie de elementos de retardo discreto o *taps*, los cuales conducen a una etapa de salida. Esta ruta, junto con lógica para detección de fase y control conforman un sistema completo con retroalimentación, tal como se muestra en la Figura 7.

La señal CLK0 es entregada a la red de distribución de señales de reloj de la FPGA, que sincroniza todo los registros del circuito que ha sido configurado. Estos registros pueden ser tanto internos como externos a la FPGA. Luego de pasar por dicha red, la señal de reloj retorna al DLL a través de la entrada CLKFB. El bloque de control del DLL mide el error de fase entre ambas señales, que es una medida del *skew* de reloj que toda la red introduce. El bloque de control activa el número apropiado de elementos de retardo para cancelar el *skew* de reloj. Una vez que se ha eliminado el desfase, se eleva la señal LOCKED, que indica la puesta en fase del reloj con respecto a la retro alimentación.

Las señales de reloj tienen una red dedicada especial para su distribución. Esta red tiene ocho entradas globales, por medio de *buffers*. La red tiene baja capacitancia y produce muy bajo *skew* de reloj, lo que la hace adecuada para

conducir señales de alta frecuencia. Tal como se muestra en la Figura 8, las entradas GCLK0 a GCLK3 están puestas en la parte inferior de la oblea de la FPGA, mientras que las entradas GCLK4 a GCLK7 están colocadas en la parte superior. Se puede conducir cualquiera de dichas entradas hacia cada uno de los CLBs, por medio de las líneas principales, que se observan en negro grueso en la Figura 8. Las líneas más delgadas representan líneas que conducen hacia los elementos síncronos de cada una de las *slices* de los CLBs.

Las entradas a la red se distribuyen a través de 4 multiplexores 2-1 a cada lado de la red, los que también conducen las señales provenientes de los DCMs. Con el propósito de minimizar la disipación de potencia dinámica en la red de distribución de relojes, el *software* de síntesis automáticamente deshabilita aquellas líneas que no son utilizadas en el diseño.

Esta red de distribución de señales de reloj es completamente independiente de la malla de interconexiones entre CLBs.

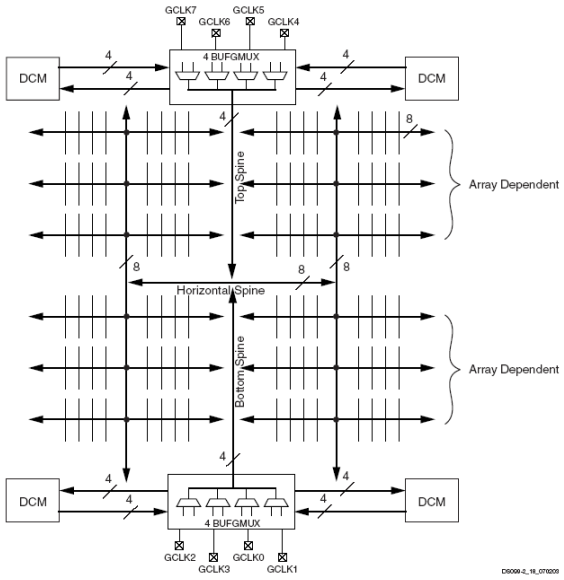


Figura 8: Red de distribución de señales de reloj de la Spartan III

Red de interconexiones de la FPGA

La red de interconexión conduce las señales entre varios elementos funcionales de la Spartan III. Hay cuatro tipos de interconexiones: *Long lines, Hex lines, Double lines y Direct lines.*

Long lines son aquellas que conectan una salida de cada seis CLBs (Figura 9a). Debido a su baja capacitancia, estas líneas son adecuadas para conducir señales de alta frecuencia. Si las ocho entradas para las redes de reloj están ocupadas, estas líneas son adecuadas como alternativa. *Hex lines* son las que conectan una salida de cada tres CLBs (Figura 9b). Son líneas que ofrecen mayor conectividad que las anteriores, pero un poco menos de capacidad en alta frecuencia. Las *Double lines* conectan todos los otros CLBs (Figura 9c), lo que las hace conexiones más flexibles. Las *Direct lines* entregan conexiones directas de cada CLB hacia cada uno de sus ocho vecinos (figura 8d). Estas líneas son usadas más a menudo para conducir una señal proveniente de un CLB de origen hacia una *Double line*, *Hex line* o *Long line* y desde esa ruta larga hacia otra *Direct line* que llevará la señal hacia el CLB de destino.

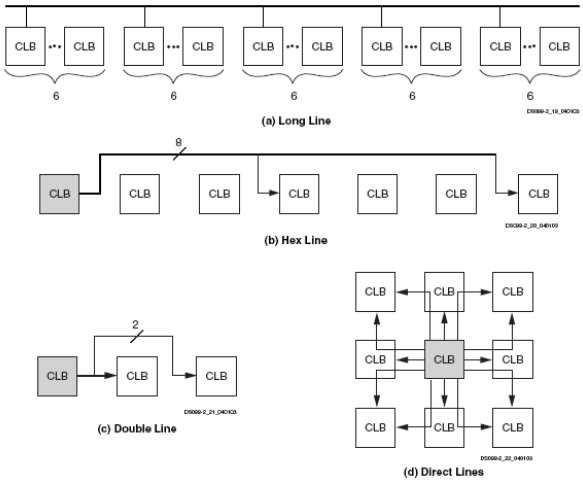


Figura 9: Tipos de interconexiones entre CLBs en la Spartan III

Proceso de configuración de la FPGA Spartan III

La FPGA Spartan III se programa por medio de la carga de los datos de configuración en celdas de memoria estática, las que colectivamente controlan todos los elementos funcionales y los recursos de interconexión. Luego de aplicar alimentación a la lámina, se escribe la trama de configuración en dicha memoria utilizando uno de los siguientes modos: Maestro - Paralelo, Esclavo - Paralelo, Maestro - Serial, Esclavo - Serial o *Boundary-Scan* (JTAG). Estos modos difieren en el origen del reloj (proviene de la FPGA en los modos Maestro y es externo en

los modos Esclavo), y en la forma en que se escriben los datos, por lo que los modos paralelos son más rápidos.

El modo *Boundary-Scan* utiliza pines dedicados de la FPGA y cumple con los estándares IEEE 1149.1 *Test Access Port* e IEEE 1532 para dispositivos *In-System Configurable* (ISC). Este modo está siempre disponible en la FPGA y al activarlo se desactivan los otros modos ya mencionados.

El proceso de configuración de la FPGA ocurre en tres etapas. Primero la memoria interna de configuración es borrada. Luego los datos de configuración son cargados en dicha memoria, y finalmente la lógica es activada por un proceso de partida.

ANEXO D

PROGRAMACION LCD

D.1 ASSEMBLER LCD

```
;*****************************************************************
;Definiciones de puertos
;*****************************************************************
            ;LCD interface ports
            ;The master enable signal is not used by the LCD display itself
            ;but may be required to confirm that LCD communication is active.
            ;This is required on the Spartan-3E Starter Kit if the StrataFLASH
            ;is used because it shares the same data pins and conflicts must be avoided.
            ;
            CONSTANT LCD_output_port, 00       ;LCD character module output data and control
            CONSTANT LCD_E, 01                 ;   active High Enable       E - bit0
            CONSTANT LCD_RW, 02        ;   Read=1 Write=0          RW - bit1
            CONSTANT LCD_RS, 04        ;   Instruction=0 Data=1    RS - bit2
            CONSTANT LCD_drive, 08     ;  Master enable (active High)-bit3--control para saber write/read
            CONSTANT LCD_DB4, 10               ;   4-bit          Data DB4 - bit4
            CONSTANT LCD_DB5, 20               ;   interface      Data DB5 - bit5
            CONSTANT LCD_DB6, 40               ;                  Data DB6 - bit6
            CONSTANT LCD_DB7, 80               ;                  Data DB7 - bit7
            ;
   NAMEREG    s6,dato1
   NAMEREG s7,dato2
   NAMEREG    s8,dig10
   NAMEREG    s9,dig32
   NAMEREG    sd,dig54
   NAMEREG    se,temp
;*********************************************************************************
;Useful data constants
;*********************************************************************************
            ;Constant to define a software delay of 1us. This must be adjusted to reflect the
            ;clock applied to KCPSM3. Every instruction executes in 2 clock cycles making the
            ;calculation highly predictable. The '6' in the following equation even allows for
            ;'CALL delay_1us' instruction in the initiating code.
            ;
            ; delay_1us_constant =  (clock_rate - 6)/4       Where 'clock_rate' is in MHz
            ;
            ;Example: For a 50MHz clock the constant value is (10-6)/4 = 11  (0B Hex).
            ;For clock rates below 10MHz the value of 1 must be used and the operation will
            ;become lower than intended.
            ;
            CONSTANT delay_1us_constant, 0B
            ;ASCII table
            CONSTANT character_a, 61
            CONSTANT character_b, 62
            CONSTANT character_c, 63
            CONSTANT character_d, 64
            CONSTANT character_e, 65
            CONSTANT character_f, 66
            CONSTANT character_g, 67
            CONSTANT character_h, 68
            CONSTANT character_i, 69
            CONSTANT character_j, 6A
            CONSTANT character_k, 6B
            CONSTANT character_l, 6C
            CONSTANT character_m, 6D
            CONSTANT character_n, 6E
            CONSTANT character_o, 6F
            CONSTANT character_p, 70
```

```
CONSTANT character_q, 71
CONSTANT character_r, 72
CONSTANT character_s, 73
CONSTANT character_t, 74
CONSTANT character_u, 75
CONSTANT character_v, 76
CONSTANT character_w, 77
CONSTANT character_x, 78
CONSTANT character_y, 79
CONSTANT character_z, 7A
CONSTANT character_A, 41
CONSTANT character_B, 42
CONSTANT character_C, 43
CONSTANT character_D, 44
CONSTANT character_E, 45
CONSTANT character_F, 46
CONSTANT character_G, 47
CONSTANT character_H, 48
CONSTANT character_I, 49
CONSTANT character_J, 4A
CONSTANT character_K, 4B
CONSTANT character_L, 4C
CONSTANT character_M, 4D
CONSTANT character_N, 4E
CONSTANT character_O, 4F
CONSTANT character_P, 50
CONSTANT character_Q, 51
CONSTANT character_R, 52
CONSTANT character_S, 53
CONSTANT character_T, 54
CONSTANT character_U, 55
CONSTANT character_V, 56
CONSTANT character_W, 57
CONSTANT character_X, 58
CONSTANT character_Y, 59
CONSTANT character_Z, 5A
CONSTANT character_0, 30
CONSTANT character_1, 31
CONSTANT character_2, 32
CONSTANT character_3, 33
CONSTANT character_4, 34
CONSTANT character_5, 35
CONSTANT character_6, 36
CONSTANT character_7, 37
CONSTANT character_8, 38
CONSTANT character_9, 39
CONSTANT character_colon, 3A
CONSTANT character_stop, 2E        ;punto
CONSTANT character_semi_colon, 3B
CONSTANT character_minus, 2D
CONSTANT character_divide, 2F      ;'/'
CONSTANT character_plus, 2B
CONSTANT character_comma, 2C
CONSTANT character_less_than, 3C
CONSTANT character_mayor, 3E    ;menor que '>'
CONSTANT character_equals, 3D
CONSTANT character_space, 20
CONSTANT character_CR, 0D          ;carriage return
CONSTANT character_question, 3F     ;'?'
```

```
            CONSTANT character_dollar, 24
            CONSTANT character_exclaim, 21      ;'!'
            CONSTANT character_BS, 08           ;Back Space command character
            CONSTANT character_teta, F2         ;teta
            ;
inicio_prog:
        CALL delay_1s
        CALL LCD_reset              ;initialise LCD display
        LOAD s5, 10                 ;Line 1 position 0
        CALL LCD_cursor
        CALL disp_EPN               ;Display 'ESC. POLITEC NAC'
        LOAD s5, 20                 ;Line 2 position 0
        CALL LCD_cursor
        CALL disp_PROYTIT           ;Display 'PROY TITILACION'
        CALL delay_1s               ;Espera 3 seg antes de cambiar de pantalla
        CALL delay_1s
        CALL delay_1s
        ;
        CALL LCD_clear              ;borrar el LCD y poner el cursor en linea 1 posicin 0
        LOAD s5,10                  ;Linea 1 posicion 0
        CALL LCD_cursor
        CALL disp_ctrlpendu         ;Muestra 'CONTROL PENDUBOT'
        LOAD s5,26                  ;Linea 2 posicion 7
        CALL LCD_cursor
        CALL disp_FPGA              ;Muestra 'FPGA'
        CALL delay_1s               ;Espera 3 seg antes de cambiar de pantalla
        CALL delay_1s
        CALL delay_1s
    call LCD_clear
Inicio:
    load s5,10
    call LCD_cursor
    call disp_ctrltope
    load s5,20
    call LCD_cursor
    call disp_ctrlmedio
        ;
;TOMAR EN CUENTA QUE PRIMERO LEE LOS PUERTOS MAS ALTOS...
prog_principal:
    input s0,00
    compare s0,01
    jump z,control_tope;
    compare s0,02
    jump z,control_tope;
    compare s0,03
    jump z,control_medio;
    compare s0,04
    jump z,control_medio;
    compare s0,00
    jump z,Inicio
    compare s0,80
    jump z,pantalla
jump prog_principal

pantalla:
      load s5,10
      call LCD_cursor
      call teta1
      call borrar
```

```
        load s5,20
        call LCD_cursor
        call teta2
        call borrar
jump prog_principal
borrar:
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        return
control_tope:
      load s5,10
      call LCD_cursor
      call disp_ctrltope
          load s5,20
          call LCD_cursor
          call teta1
          load s5,28
          call LCD_cursor
          call teta2
      jump prog_principal

control_medio:
      load s5,10
      call LCD_cursor
      call disp_ctrlmedio
          load s5,20
          call LCD_cursor
          call teta1
          load s5,28
          call LCD_cursor
          call teta2
      jump prog_principal

teta1:   LOAD s5, character_teta
         CALL LCD_write_data              ;Muestra teta1
         LOAD s5, character_1
         CALL LCD_write_data
         LOAD s5, character_equals
         CALL LCD_write_data
         input s0,01
         and s0,01
         compare s0,01
         call z,negativo
```

```
        input sa,02
        input sb,03
        call binario_bcd
        call disp_numero
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        return
        ;
teta2:   LOAD s5, character_teta
        CALL LCD_write_data                ;Muestra teta2
        LOAD s5, character_2
        CALL LCD_write_data
        LOAD s5, character_equals
        CALL LCD_write_data
        input s0,01
        and s0,02
        compare s0,02
        call z,negativo
        input sa,04
        input sb,05
        call binario_bcd
        call disp_numero
        LOAD s5, character_space
        CALL LCD_write_data
        LOAD s5, character_space
        CALL LCD_write_data
        return
        ;
negativo:
       LOAD s5, character_minus
       CALL LCD_write_data                 ;Muestra menu 2
       return
;***************************************************************************************
;Software delay routines
;***************************************************************************************
;Delay of 1us.
;
;Constant value defines reflects the clock applied to KCPSM3. Every instruction
;executes in 2 clock cycles making the calculation highly predictable. The '6' in
;the following equation even allows for 'CALL delay_1us' instruction in the initiating code.
;
; delay_1us_constant =  (clock_rate - 6)/4       Where 'clock_rate' is in MHz
;
;Registers used s0
;
     delay_1us: LOAD s0, delay_1us_constant
      wait_1us: SUB s0, 01
                JUMP NZ, wait_1us
                RETURN
                ;
                ;Delay of 40us.
                ;
                ;Registers used s0, s1
                ;
    delay_40us: LOAD s1, 28                 ;40 x 1us = 40us
     wait_40us: CALL delay_1us
                SUB s1, 01
```

```
            JUMP NZ, wait_40us
            RETURN
            ;
            ;Delay of 1ms.
            ;
            ;Registers used s0, s1, s2
            ;
     delay_1ms: LOAD s2, 19                 ;25 x 40us = 1ms
      wait_1ms: CALL delay_40us
            SUB s2, 01
            JUMP NZ, wait_1ms
            RETURN
            ;
            ;Delay of 20ms.
            ;
            ;Delay of 20ms used during initialisation.
            ;
            ;Registers used s0, s1, s2, s3
            ;
    delay_20ms: LOAD s3, 14                 ;20 x 1ms = 20ms
     wait_20ms: CALL delay_1ms
            SUB s3, 01
            JUMP NZ, wait_20ms
            RETURN
            ;
            ;Delay of approximately 1 second.
            ;
            ;Registers used s0, s1, s2, s3, s4
            ;
      delay_1s: LOAD s4, 32                 ;50 x 20ms = 1000ms
       wait_1s: CALL delay_20ms
            SUB s4, 01
            JUMP NZ, wait_1s
            RETURN
;***************************************************************************************
;LCD Character Module Routines
;***************************************************************************************
;
;LCD module is a 16 character by 2 line display but all displays are very similar
;The 4-wire data interface will be used (DB4 to DB7).
;
;The LCD modules are relatively slow and software delay loops are used to slow down
;KCPSM3 adequately for the LCD to communicate. The delay routines are provided in
;a different section (see above in this case).
;
;Pulse LCD enable signal 'E' high for greater than 230ns (1us is used).
;
;Register s4 should define the current state of the LCD output port.
;Registers used s0, s4
;
   LCD_pulse_E: XOR s4, LCD_E               ;E=1
            OUTPUT s4, LCD_output_port
            CALL delay_1us
            XOR s4, LCD_E               ;E=0
            OUTPUT s4, LCD_output_port
            RETURN
            ;
            ;Write 4-bit instruction to LCD display.
            ;
```

```
            ;The 4-bit instruction should be provided in the upper 4-bits of register s4.
            ;Note that this routine does not release the master enable but as it is only
            ;used during initialisation and as part of the 8-bit instruction write it
            ;should be acceptable.
            ;
            ;Registers used s4
            ;
LCD_write_inst4: AND s4, F8                 ;Enable=1 RS=0 Instruction, RW=0 Write, E=0
            OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns before enable pulse
            CALL LCD_pulse_E
            RETURN
            ;
            ;Write 8-bit instruction to LCD display.
            ;The 8-bit instruction should be provided in register s5.
            ;Instructions are written using the following sequence
            ; Upper nibble
            ; wait >1us
            ; Lower nibble
            ; wait >40us
            ;
            ;Registers used s0, s1, s4, s5
            ;
LCD_write_inst8: LOAD s4, s5
            AND s4, F0                  ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
            OR s4, LCD_drive            ;Enable=1
            CALL LCD_write_inst4        ;write upper nibble
            CALL delay_1us              ;wait >1us
            LOAD s4, s5                 ;select lower nibble with
            SL1 s4                      ;Enable=1
            SL0 s4                      ;RS=0 Instruction
            SL0 s4                      ;RW=0 Write
            SL0 s4                      ;E=0
            CALL LCD_write_inst4        ;write lower nibble
            CALL delay_40us             ;wait >40us
            LOAD s4, F0                 ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
            OUTPUT s4, LCD_output_port  ;Release master enable
            RETURN
            ;
            ;Write 8-bit data to LCD display.
            ;
            ;The 8-bit data should be provided in register s5.
            ;Data bytes are written using the following sequence
            ; Upper nibble
            ; wait >1us
            ; Lower nibble
            ; wait >40us
            ;
            ;Registers used s0, s1, s4, s5
            ;
 LCD_write_data: LOAD s4, s5
            AND s4, F0                  ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
            OR s4, 0C                   ;Enable=1 RS=1 Data, RW=0 Write, E=0
            OUTPUT s4, LCD_output_port  ;set up RS and RW >40ns before enable pulse
            CALL LCD_pulse_E            ;write upper nibble
            CALL delay_1us              ;wait >1us
            LOAD s4, s5                 ;select lower nibble with
            SL1 s4                      ;Enable=1
            SL1 s4                      ;RS=1 Data
            SL0 s4                      ;RW=0 Write
```

```
        SL0 s4                      ;E=0
        OUTPUT s4, LCD_output_port      ;set up RS and RW >40ns before enable pulse
        CALL LCD_pulse_E            ;write lower nibble
        CALL delay_40us             ;wait >40us
        LOAD s4, F0                 ;Enable=0 RS=0 Instruction, RW=0 Write, E=0
        OUTPUT s4, LCD_output_port      ;Release master enable
        RETURN
        ;
        ;
        ;Reset and initialise display to communicate using 4-bit data mode
        ;Includes routine to clear the display.
        ;
        ;Requires the 4-bit instructions 3,3,3,2 to be sent with suitable delays
        ;following by the 8-bit instructions to set up the display.
        ;
        ;  28 = '001' Function set, '0' 4-bit mode, '1' 2-line, '0' 5x7 dot matrix, 'xx'
        ;  06 = '000001' Entry mode, '1' increment, '0' no display shift
        ;  0C = '00001' Display control, '1' display on, '0' cursor off, '0' cursor blink off
        ;  01 = '00000001' Display clear
        ;
        ;Registers used s0, s1, s2, s3, s4
        ;
LCD_reset: CALL delay_20ms              ;wait more that 15ms for display to be ready
        LOAD s4, 30
        CALL LCD_write_inst4        ;send '3'
        CALL delay_20ms             ;wait >4.1ms
        CALL LCD_write_inst4        ;send '3'
        CALL delay_1ms              ;wait >100us
        CALL LCD_write_inst4        ;send '3'
        CALL delay_40us             ;wait >40us
        LOAD s4, 20
        CALL LCD_write_inst4        ;send '2'
        CALL delay_40us             ;wait >40us
        LOAD s5, 28                 ;Function set
        CALL LCD_write_inst8
        LOAD s5, 06                 ;Entry mode
        CALL LCD_write_inst8
        LOAD s5, 0C                 ;Display control
        CALL LCD_write_inst8
LCD_clear: LOAD s5, 01                  ;Display clear
        CALL LCD_write_inst8
        CALL delay_1ms              ;wait >1.64ms for display to clear
        CALL delay_1ms
        RETURN
        ;
        ;Position the cursor ready for characters to be written.
        ;The display is formed of 2 lines of 16 characters and each
        ;position has a corresponding address as indicated below.
        ;
        ;               Character position
        ;         0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
        ;
        ; Line 1 - 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
        ; Line 2 - C0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
        ;
        ;This routine will set the cursor position using the value provided
        ;in register s5. The upper nibble will define the line and the lower
        ;nibble the character position on the line.
        ; Example s5 = 2B will position the cursor on line 2 position 11
```

```
            ;
            ;Registers used s0, s1, s2, s3, s4
            ;
    LCD_cursor: TEST s5, 10                 ;test for line 1
            JUMP Z, set_line2
            AND s5, 0F                  ;make address in range 80 to 8F for line 1
            OR s5, 80
            CALL LCD_write_inst8            ;instruction write to set cursor
            RETURN
     set_line2: AND s5, 0F                  ;make address in range C0 to CF for line 2
            OR s5, C0
            CALL LCD_write_inst8            ;instruction write to set cursor
            RETURN
;*****************************************************************************************
;LCD text messages
;*****************************************************************************************
;Display 'ESC. POLITEC NAC' on LCD at current cursor position
            ;
   disp_EPN:    LOAD s5, character_E
            CALL LCD_write_data
            LOAD s5, character_S
            CALL LCD_write_data
            LOAD s5, character_C
            CALL LCD_write_data
            LOAD s5, character_stop
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_P
            CALL LCD_write_data
            LOAD s5, character_O
            CALL LCD_write_data
            LOAD s5, character_L
            CALL LCD_write_data
            LOAD s5, character_I
            CALL LCD_write_data
            LOAD s5, character_T
            CALL LCD_write_data
            LOAD s5, character_E
            CALL LCD_write_data
            LOAD s5, character_C
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_N
            CALL LCD_write_data
            LOAD s5, character_A
            CALL LCD_write_data
            LOAD s5, character_C
            CALL LCD_write_data
            RETURN
            ;
             ;Display '>PROY TITULACION' on LCD at current cursor position
            ;
     disp_PROYTIT:  LOAD s5, character_P
                CALL LCD_write_data
                LOAD s5, character_R
                CALL LCD_write_data
                LOAD s5, character_O
```

```
            CALL LCD_write_data
            LOAD s5, character_Y
            CALL LCD_write_data
            LOAD s5, character_stop
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_T
            CALL LCD_write_data
            LOAD s5, character_I
            CALL LCD_write_data
            LOAD s5, character_T
            CALL LCD_write_data
            LOAD s5, character_U
            CALL LCD_write_data
            LOAD s5, character_L
            CALL LCD_write_data
            LOAD s5, character_A
            CALL LCD_write_data
            LOAD s5, character_C
            CALL LCD_write_data
            LOAD s5, character_I
            CALL LCD_write_data
            LOAD s5, character_O
            CALL LCD_write_data
            LOAD s5, character_N
            CALL LCD_write_data
            RETURN
            ;
            ;muestra "CONTROL PENDUBOT" en la actual posicion del cursor en el LCD
            ;
disp_ctrlpendu:   LOAD s5, character_C
              CALL LCD_write_data
              LOAD s5, character_O
              CALL LCD_write_data
              LOAD s5, character_N
              CALL LCD_write_data
              LOAD s5, character_T
              CALL LCD_write_data
              LOAD s5, character_R
              CALL LCD_write_data
              LOAD s5, character_O
              CALL LCD_write_data
              LOAD s5, character_L
              CALL LCD_write_data
              LOAD s5, character_space
              CALL LCD_write_data
              LOAD s5, character_P
              CALL LCD_write_data
              LOAD s5, character_E
              CALL LCD_write_data
              LOAD s5, character_N
              CALL LCD_write_data
              LOAD s5, character_D
              CALL LCD_write_data
              LOAD s5, character_U
              CALL LCD_write_data
              LOAD s5, character_B
              CALL LCD_write_data
```

```
            LOAD s5, character_O
            CALL LCD_write_data
            LOAD s5, character_T
            CALL LCD_write_data
            RETURN
            ;
        ;muestra "FPGA" en la actual posicion del cursor en el LCD
        ;
disp_FPGA:  LOAD s5, character_F
        CALL LCD_write_data
        LOAD s5, character_P
        CALL LCD_write_data
        LOAD s5, character_G
        CALL LCD_write_data
        LOAD s5, character_A
        CALL LCD_write_data
        RETURN
        ;
        ;muestra ">Control Tope" en la actual posicion del cursor en el LCD
        ;
disp_ctrltope:  LOAD s5, character_mayor
            CALL LCD_write_data
            LOAD s5, character_C
            CALL LCD_write_data
            LOAD s5, character_t
            CALL LCD_write_data
            LOAD s5, character_r
            CALL LCD_write_data
            LOAD s5, character_l
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_T
            CALL LCD_write_data
            LOAD s5, character_o
            CALL LCD_write_data
            LOAD s5, character_p
            CALL LCD_write_data
            LOAD s5, character_e
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, 7E          ;flecha derecha
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            LOAD s5, character_space
            CALL LCD_write_data
            RETURN
            ;

            ;muestra ">Control Tope" en la actual posicion del cursor en el LCD
            disp_ctrlmedio:  LOAD s5, character_mayor
            CALL LCD_write_data
            LOAD s5, character_C
```

```
                CALL LCD_write_data
                LOAD s5, character_t
                CALL LCD_write_data
                LOAD s5, character_r
                CALL LCD_write_data
                LOAD s5, character_l
                CALL LCD_write_data
                LOAD s5, character_space
                CALL LCD_write_data
                LOAD s5, character_M
                CALL LCD_write_data
                LOAD s5, character_e
                CALL LCD_write_data
                LOAD s5, character_d
                CALL LCD_write_data
                LOAD s5, character_i
                CALL LCD_write_data
                LOAD s5, character_o
                CALL LCD_write_data
                LOAD s5, character_space
                CALL LCD_write_data
                LOAD s5, character_space
                CALL LCD_write_data
                LOAD s5, 7F       ;flecha izquierda
                CALL LCD_write_data
                LOAD s5, character_space
                CALL LCD_write_data
                LOAD s5, character_space
                CALL LCD_write_data
                RETURN
disp_numero:
        LOAD s5, dig54
        CALL LCD_write_data
        LOAD s5, dig32
        CALL LCD_write_data
        LOAD s5, dig10
        CALL LCD_write_data
        RETURN
;*****************************************************************************************
;DE BINARIO A BCD DE 16 BITS
;*****************************************************************************************
;Convierte una entrada binaria de 16 bits a 5 num en BCD
binario_bcd:
          load dig10,00
          load dig32,00
          load dig54,00

          load dato1,sa
          load dato2,sb
          and sb,0F
          load sf,10        ;16 bits
next_bit:
          sl0 dato1                 ;desplazamiento a la izquierda
          sla dato2
          sla dig10                 ;desplazamiento a la izquierda con carry
          sla dig32                 ;desplazamiento a la izquierda con carry
   sla dig54
          sub sf,01
          jump z,fin
```

```
    load temp,03
            add temp,dig10
            test temp,08
            jump z,salto1
    load dig10,temp
salto1:
    load temp,30
    add temp,dig10
    test temp,80
    jump z,salto2
    load dig10,temp
salto2:
    load temp,03
            add temp,dig32
            test temp,08
            jump z,salto3
    load dig32,temp
 salto3:
    load temp,30
    add temp,dig32
    test temp,80
    jump z,salto4
    load dig32,temp
salto4:
    load temp,03
            add temp,dig54
            test temp,08
            jump z,salto5
    load dig54,temp
salto5:
    load temp,30
    add temp,dig54
    test temp,80
    jump z,salto6
    load dig54,temp
salto6:
   jump next_bit
fin:
    load dig54,dig32
    load dato1,dig32
    load dig32,dig10
    and dig10,0f
    sr0 dig32
    sr0 dig32
    sr0 dig32
    sr0 dig32
    and dig54,0f
    sr0 dato1
    sr0 dato1
    sr0 dato1
    sr0 dato1
    add dig10,30            ;digito 0
    add dig32,30            ;digito 1
    add dig54,30            ;digito 2
    add dato1,30            ;digito 3
    return
```

D.2 PROGRAMA LCD VHDL

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

entity control_lcd is
   Port (       clk : in std_logic;
                              reset_LCD : in std_logic;
               strataflash_oe : out std_logic;
                        strataflash_we : out std_logic;
                                                lcd_d : out std_logic_vector(3 downto 0);
                                       lcd_rs : out std_logic;
                                       lcd_rw : out std_logic;
                                                lcd_e : out std_logic;
                                      pantalla: in std_logic;
                                control  : in std_logic_vector (7 downto 0);
                                        teta1                : in std_logic_vector (9 downto 0);
                                        teta2                : in std_logic_vector (9 downto 0));
end control_lcd;

architecture Behavioral of control_lcd is
----------------------------------------------------------------------------------
-- declaration of KCPSM3
----------------------------------------------------------------------------------
COMPONENT kcpsm3
          PORT(
                    instruction : IN std_logic_vector(17 downto 0);
                    in_port : IN std_logic_vector(7 downto 0);
                    interrupt : IN std_logic;
                    reset : IN std_logic;
                    clk : IN std_logic;
                    address : OUT std_logic_vector(9 downto 0);
                    port_id : OUT std_logic_vector(7 downto 0);
                    write_strobe : OUT std_logic;
                    out_port : OUT std_logic_vector(7 downto 0);
                    read_strobe : OUT std_logic;
                    interrupt_ack : OUT std_logic);
END COMPONENT;
----------------------------------------------------------------------------------
-- declaration of program ROM
----------------------------------------------------------------------------------
COMPONENT lcd_disp
          PORT(
                    address : IN std_logic_vector(9 downto 0);
                    clk : IN std_logic;
                    instruction : OUT std_logic_vector(17 downto 0));
END COMPONENT;
----------------------------------------------------------------------------------
-- Signals used to connect KCPSM3 to program ROM and I/O logic
----------------------------------------------------------------------------------
          signal   instruction          : std_logic_vector(17 downto 0);
          signal    in_port             : std_logic_vector(7 downto 0);
          signal    interrupt: std_logic:='0';              --no se necesita interrupcion
          signal    reset                         : std_logic:= '0';                      --reset en false
          signal    address             : std_logic_vector(9 downto 0);
          signal    port_id             : std_logic_vector(7 downto 0);
          signal    write_strobe: std_logic;
          signal    out_port            : std_logic_vector(7 downto 0);
```

```
            signal    read_strobe         : std_logic;
            signal    interrupt_ack: std_logic;
----------------------------------------------------------------------------------
-- Señales para la operacion del LCD
----------------------------------------------------------------------------------
            signal   lcd_rw_control : std_logic;
            signal  lcd_output_data : std_logic_vector(7 downto 4);
            signal        lcd_drive : std_logic;
begin
----------------------------------------------------------------------------------
-- Deshabilitacion de los componenetes que son compartidos y que no se usan
----------------------------------------------------------------------------------
--StrataFLASH must be disabled to prevent it conflicting with the LCD display
  strataflash_oe <= '1';
  strataflash_we <= '1';
----------------------------------------------------------------------------------
-- Control del LCD
----------------------------------------------------------------------------------
--Control of read and write signal
  lcd_rw <= lcd_rw_control and lcd_drive;
--use read/write control to enable output buffers.
  lcd_d <= lcd_output_data when (lcd_rw_control='0' and lcd_drive='1') else "ZZZZ";
----------------------------------------------------------------------------------
-- Instanciacion del KCPSM3 y el programa de memoria
----------------------------------------------------------------------------------
--Microprocesador
lcd_kcpsm3: kcpsm3 PORT MAP(
                        address => address,
                        instruction => instruction ,
                        port_id => port_id,
                        write_strobe => write_strobe,
                        out_port => out_port,
                        read_strobe => read_strobe,
                        in_port => in_port,
                        interrupt => interrupt,
                        interrupt_ack => interrupt_ack,
                        reset => reset,
                        clk => clk
            );
--Memoria lcd
lcd: lcd_disp PORT MAP(
                        address => address,
                        instruction => instruction,
                        clk => clk
            );
----------------------------------------------------------------------------------
-- KCPSM3 output ports
----------------------------------------------------------------------------------
output_ports: process(clk)
  begin
    if clk'event and clk='1' then
      if write_strobe='1' then
        if port_id(0)='0' then
          lcd_output_data <= out_port(7 downto 4);
          lcd_drive <= out_port(3);
          lcd_rs <= out_port(2);
          lcd_rw_control <= out_port(1);
          lcd_e <= out_port(0);
        end if;
```

```
     end if;
   end if;
end process;
--------------------------------------------------------------------------------
-- KCPSM3 input ports
--------------------------------------------------------------------------------
input_ports: process(clk)
          variable t1_aux:std_logic_vector(9 downto 0):="0000000000";
          variable t2_aux:std_logic_vector(9 downto 0):="0000000000";
          variable signo_t1:std_logic:='0';
          variable signo_t2:std_logic:='0';
          variable control_aux:std_logic_vector(7 downto 0);
begin
          if clk'event and clk='1' then
-- Deteccion de signos
                    if teta1(9)='1' then
                              signo_t1:=teta1(9);
                              t1_aux:=not teta1;
                              t1_aux:=t1_aux+1;
                    else
                              signo_t1:='0';
                              t1_aux:=teta1;
                    end if;

                    if teta2(9)='1' then
                              signo_t2:=teta2(9);
                              t2_aux:=not teta2;
                              t2_aux:=t2_aux+1;
                    else
                              signo_t2:='0';
                              t2_aux:=teta2;
                    end if;
--Envio de datos
                    control_aux:=pantalla&control(6 downto 0);
                    case port_id(3 downto 0) is
                              when "0000"  =>  in_port<=control_aux;     --tipo de control
                              when "0001"  =>  in_port<="000000"&signo_t2&signo_t1;   --signos
                              when "0010"  =>  in_port<=t1_aux(7 downto 0);                --teta1
                              when "0011"  =>  in_port<="000000"&t1_aux(9 downto 8);
                              when "0100"  =>  in_port<=t2_aux(7 downto 0);                --teta2
                              when "0101"  =>  in_port<="000000"&t2_aux(9 downto 8);
                              when others  =>  in_port<="XXXXXXXX";
                    end case;
           end if;
end process input_ports;

end Behavioral;
```