UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA DE INGENIERÍA ELÉCTRICA

Diseño de un controlador PID, utilizando una tarjeta FPGA cyclone V GX starter, programada en VHDL.

PRESENTADO POR:

JOSÉ RAÚL MEJÍA NUILA

GUILLERMO ALFONSO MAXIMILIANO NARVÁEZ HENRÍQUEZ

PARA OPTAR AL TÍTULO DE:

INGENIERO ELECTRICISTA

CIUDAD UNIVERSITARIA, JULIO DE 2016

UNIVERSIDAD DE EL SALVADOR

RECTOR INTERINO :

LIC. JOSÉ LUIS ARGUETA ANTILLÓN

SECRETARIA GENERAL :

DRA. ANA LETICIA ZAVALETA DE AMAYA

FACULTAD DE INGENIERÍA Y ARQUITECTURA

DECANO :

ING. FRANCISCO ANTONIO ALARCÓN SANDOVAL

SECRETARIO :

ING. JULIO ALBERTO PORTILLO

ESCUELA DE INGENIERÍA ELÉCTRICA

DIRECTOR :

ING. ARMANDO MARTÍNEZ CALDERÓN

UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA DE INGENIERÍA ELÉCTRICA

Trabajo de Graduación previo a la opción al Grado de:

INGENIERO ELECTRICISTA

Título :

Diseño de un controlador PID, utilizando una tarjeta FPGA cyclone V GX starter, programada en VHDL.

Presentado por :

JOSÉ RAÚL MEJÍA NUILA

GUILLERMO ALFONSO MAXIMILIANO NARVÁEZ HENRÍQUEZ

Trabajo de Graduación Aprobado por :

Docente Asesor :

MSC. SALVADOR DE JESÚS GERMÁN

San Salvador, julio de 2016

Trabajo de Graduación Aprobado por:

Docente Asesor :

MSC. SALVADOR DE JESÚS GERMÁN

ACTA DE CONSTANCIA DE NOTA Y DEFENSA FINAL

En esta fecha, Martes 19 de julio de 2016, en la Sala de Lectura de la Escuela de Ingeniería Eléctrica, a las 10:00 a.m. horas, en presencia de las siguientes autoridades de la Escuela de Ingeniería Eléctrica de la Universidad de El Salvador:

1. Ing. Armando Martínez Calderón
   Director — Firma:

2. MSc. José Wilber Calderón Urrutia
   Secretario — Firma: Wilber Calderón

Y, con el Honorable Jurado de Evaluación integrado por las personas siguientes:

1- MSc. Salvador de Jesús German — Firma:

2- MSc. José Wilber Calderón Urrutia — Wilber Calderón

3- MSc. Ricardo Ernesto Cortez

Se efectuó la defensa final reglamentaria del Trabajo de Graduación:

Diseño de un controlador PID, utilizando una tarjeta FPGA cyclone V GX starter, programada en VHDL.

A cargo de los Bachilleres:

- Mejía Nuila José Raúl

- Narváez Henríquez Guillermo Alfonso Maximiliano

Habiendo obtenido en el presente Trabajo una nota promedio de la defensa final: 8.2

(ocho . dos)

Agradecimientos.

Este es final de un largo trayecto que sin la bendición y ayuda de Dios no hubiera podido alcanzar. Gracias Dios mío por regalarme mi familia que siempre me ha dado su apoyo incondicional, las palabras de aliento de cada uno, los ejemplos de ser personas de bien y su disposición a brindarme una mano siempre que los necesito. A mi madre Nelly Henríquez un agradecimiento especial por su eterno sacrificio, su lucha para convertirme un hombre de bien y regalarme una vida profesional, por mantener siempre su presión para guiarme por el mejor curso de la vida. A mi padre José Maximiliano Narváez que en paz descanse, quien con sus enseñanzas y deseos de superación para mí, convirtieron en promesas nuestros sueños, gracias papá.

Gracias a mis hermanos Carlos Narváez Y Rosa Narváez, primos, cuñados, pareja, los que siempre han estado pendiente de mí y han sido un ejemplo que con sus consejos y apoyo han evitado que los embates de la vida me golpeen y me hagan desistir. A mis tías que siempre me dieron su apoyo para fomentar mi educación, que siempre estuvieron pendientes de mí y pusieron toda su disposición para ayudarme a lograr mis metas. A mis amigos, que estuvieron a lo largo de este trayecto para compartir éxitos y fracasos, gracias.

A nuestro Asesor, Ing. German Salvador. Gracias por su paciencia, consejos, apoyo y los conocimientos que compartió para lograr culminar este trabajo.

Al resto de mi familia que su mención he quedado debiendo pero que han sido parte de este éxito, a las personas que se adelantaron en el camino de la vida y que en la tierra y desde el cielo nos dieron su ayuda, gracias totales.

Maximiliano Narváez

Agradecimientos.

Al llegar al final de mi carrera universitaria es imprescindible agradecer a mi Dios, siempre está y estará, ya sea en las buenas y en las malas, su ayuda su fortaleza, su sabiduría, me dio la fuerza necesaria para culminar este éxito.

A mis padres Raúl Antonio Mejía y a mi madre Gloria Luz Nuila de Mejía (Q.E.P.D) que sin su apoyo, sus consejos y paciencia hubiese sido casi imposible llegar al final de esta exitosa carrera, pues, yo hice los más fácil, que es, estudiar, lo más difícil lo hicieron ellos, ese apoyo económico ese arduo trabajo que sin cansancio y mucho esfuerzo me han servido de ejemplo para nunca en la vida rendirme a pesar de las situaciones que la vida enfrente.

A mis hermanos Jacqueline Mejía y Daniel Mejía que, igual a mis padres siempre estuvieron en los momentos que parecía desfallecer y sus palabras de ánimo me empujaron para seguir adelante y culminar este triunfo.

A una gran persona Álvaro Ernesto Nuila, que en un momento tan difícil ofreció su ayuda sin dudar, sin esperar nada a cambio, me ha servido de ejemplo en la vida, para darme cuenta que con pequeñas acciones y con el servicio a las demás personas se puede cambiar el mundo.

A mis amigos que igual que las demás personas, siempre me dieron sus buenos deseos y su fuerza de ánimo para llegar al final.

A mis compañeros, grandes personas, compartimos tristezas, alegrías, fracasos y sobre todo muchos triunfos, esperando que, esos triunfos, también los compartamos en nuestra vida profesional.

A toda la familia Mejía y a toda la familia Nuila, muchas gracias, siempre estuvieron pendientes, de una u otra manera, e igual a todas las buenas personas deseaban con mucho ánimo el culminar de esta carrera.

A mi asesor de tesis Ing. Salvador de Jesús German, gracias por tan grande labor, gracias por su paciencia, gracias por su apoyo y por brindar sus enormes conocimientos que compartió para llegar a culminar este trabajo.

A todas aquellas personas que no se mencionan y que siempre brindaron su apoyo, sus muestras de ánimo y fuerzas para continuar, muchas gracias.

Y a las personas que partieron en el camino de la vida y que desde el cielo ven este gran triunfo con mucha alegría, también es de ellos, gracias.

José Raúl Mejía Nuila.

Dedicatoria

A Gloria Luz Nuila de Mejía, por tan hermoso ser que Dios utilizó para
darme la vida, deseando con todo corazón que estuviese acá,
ella, aunque no obtuvo nunca un título profesional, más que el mejor título
que Dios le brindo, el ser la mejor madre,
ella es una ingeniera eléctrica.
Te amare por siempre.

José Raúl Mejía Nuila.

Contenido

Índice de figuras

Índice de tablas

Índice de ecuaciones

Introducción.

Las FPGA (del inglés *Field Programmable Gate Array*) son dispositivos que tienen un gran número de elementos lógicos programables, esta tecnología con anterioridad se limitaba a ingenieros con un amplio conocimiento sobre hardware digital, en la actualidad el avance en los programas de descripción de hardware (HDL) y la disponibilidad de éstas en el mercado han hecho que su uso prolifere en distintas áreas de la industria, como ejemplo en sistemas de seguridad, aeronáutica, robótica, automatización y en general todo sistema digital.

Los dispositivos FPGA al ser programados se convierten en verdadero hardware, esto los diferencia de otros dispositivos programables ya que las instrucciones programadas ocurren de forma concurrente y no de forma secuencial esto quiere decir que los cambios en las señales conllevan a cambios inmediatos en los nodos del sistema digital y esto se transforma en alta velocidad de procesamiento.

Altera es una compañía estandarte en la fabricación de chips FPGA, además de ofrecer tarjetas que integran diversos periféricos conjuntos al chip FPGA, proporcionan las herramientas necesarias para diversos tipos de proyecto, brindan el software (una versión gratuita) necesario para el diseño y simulación del hardware, este trabajo en particular utiliza la tarjeta cyclone V GX starter de Altera junto al software Quartus II y Modelsim para el diseño e implementación.

La inclusión de dispositivos FPGA a procesos de automatización tiene sus razones en la capacidad del dispositivo en atender de forma paralela distintos procesos sin restarse recursos el uno del otro, la velocidad de procesamiento, el manejo de cadenas de bit (como lo haría un PLC) y un gran número de puertos de entrada/salida para la recepción y envío de señales que en muchos otros dispositivos se compran de manera separada.

En diferentes tipos de entorno se encuentran dispositivos encargados de ejercer control sobre los equipos que ejecutan tareas de un proceso, dichos dispositivos pueden estar en el diseño interno o externos y se denominan “controladores”, algunos ejemplos de control llevados a cabo con estos son : control de temperatura, control de presión, control de velocidad, control de posición. La elección del tipo de controlador depende de las características físicas del sistema y su fin.

En este trabajo se pretende generalizar mediante el diseño de un controlador PID (acrónimo de: Proporcional, Integral, Derivativo) el uso de la tarjeta cyclone V GX starter debido a que el controlador PID es tradicionalmente uno de los más usados para diferentes tipos de control. Para llevar a cabo la tarea del diseño hay que incurrir en los aspectos básicos de la teoría de control, y conocer los elementos de hardware para llevar a cabo su prueba. Para la aplicación del controlador PID se ha propuesto ejercer control de posición y velocidad sobre un motor de corriente continua.

A lo largo de los capítulos subsiguientes se conocen los elementos necesarios para llevar a cabo el sistema de control, desde la teoría de control, la descripción del hardware a usar, el software de programación / simulación y el software de presentación de datos.

Objetivos.

Objetivo general:

- Diseñar e implementar un controlador proporcional, integral y derivativo (PID) digital en la tarjeta FPGA cyclone V GX starter KIT de Altera usando el lenguaje de programación VHDL, y aplicarlo al control de posición y velocidad de un motor de corriente continua.

Objetivos específicos:

- Utilizar las herramientas de programación del lenguaje VHDL para diseñar un hardware que funcione como un controlador PID digital y de esta forma usar la tarjeta cyclone V como una herramienta para procesos que requieran ese tipo de controlador.

- Reducir el error entre una entrada de referencia y una señal de realimentación mediante un controlador PID digital acoplando la salida del PID a la señal actuante del sistema.

- Utilizar programación jerárquica en VHDL para dividir el controlador PID digital en componentes, manteniendo integro el componente PID para ser usado tanto en control de posición y control de velocidad para un motor de corriente continua.

- Sintonizar el controlador PID para obtener una respuesta satisfactoria del sistema manejando los parámetros influyentes del controlador PID (las constantes proporcional, integral y derivativa) para ejercer efecto en la respuesta transitoria y de estado estable del sistema a controlar.

- Graficar la repuesta del sistema ante una entrada escalón utilizando la transmisión de datos por medio del puerto USB de la tarjeta Cyclone V y el software Labview.

Capítulo I: Teoría de control para el diseño del controlador PID.

Introducción al capítulo I.

Para el diseño de un controlador PID es necesario conocer aspectos y conceptos de la teoría de control automático. En el presente capítulo se presenta la teoría necesaria para la implementación de un controlador PID digital ya que se incurre a temas como la respuesta del sistema en el tiempo, el traspaso del algoritmo PID para tiempo continuo a tiempo discreto, los efectos de las acciones proporcional, integral y derivativa sobre la acción total del controlador entre otros puntos importantes a tomar en cuenta en el diseño.

La aplicación del controlador PID se lleva a cabo controlando la posición y velocidad de un motor de corriente continua cuya característica más relevante es tener acoplado un encoder incremental a su eje, por tal es necesario introducir los conceptos necesarios para utilizar este hardware y lograr el movimiento controlado del motor.

Ya que se pretende evaluar la salida del sistema (la posición o velocidad del eje del motor) a medida transcurre el tiempo es necesario establecer la forma de comunicación que tendrá la tarjeta FPGA con la computadora, una vez establecida esta forma de comunicación se procede a la elección del software para presentación de resultados, para el caso se ha seleccionado el programa Labview.

1.1 Respuesta de sistemas en el tiempo.

Cuando se diseña un sistema de control, se evalúa su comportamiento a medida que el tiempo transcurre, este hecho hace que la variable independiente de muchos sistemas sea el tiempo, esto no es excluyente en el caso del control de posición y velocidad del motor dc ya que si bien el resultado final es una posición o velocidad deseada es importante del cómo se llega a ese resultado. normalmente la respuesta en el tiempo se divide en dos partes, una que se extingue a medida que el tiempo se vuelve grande que se denomina respuesta transitoria y otra que prevalece en el tiempo que es la respuesta en estado estable.

Una representación matemática de la respuesta transitoria y de estado estable se muestra en la ecuación 1.1

$$y(t) = y_t(t) + y_{ss}(t) \qquad \textit{Ecuación 1.1}$$

En donde $y_t(t)$ indica la respuesta transitoria y $y_{ss}(t)$ la respuesta en estado estable.

Ya que la respuesta transitoria se extingue a medida que el tiempo se hace grande (tiende a infinito) esto se describe mediante la Ecuación 1.2:

$$\lim_{t\to\infty} y_t(t) = 0 \qquad \textit{Ecuación 1.2}$$

Es importante mencionar que todos los sistemas de control reales presentan una respuesta transitoria ya que la respuesta de un sistema no puede seguir la referencia de una forma súbita (la salida siguiendo a la entrada).

Tanto para el análisis de la respuesta transitoria y de estado estable existen parámetros que identifican su comportamiento y métodos para acoplar estos parámetros para cumplir los requerimientos de un sistema particular, es importante por ejemplo conocer la respuesta transitoria para limitar la desviación de la salida con respecto a una entrada y la respuesta en estado estable para conocer la exactitud final del sistema.

En general cuando la respuesta en estado estable no concuerda con la referencia, entonces el sistema tiene un error de estado estable (ess).

1.1.1 Señales típicas para obtener la respuesta en el tiempo de sistemas de control.

Para conocer la respuesta en el tiempo de un sistema de control se usan señales de entrada que evalúan distintas situaciones a las cuales puede ser sometido el sistema, con estas entradas se obtienen parámetros que permiten analizar, predecir y corregir el funcionamiento de este además de sistematizar el tratamiento matemático y predecir el funcionamiento de este ante entradas más complejas.
Particularmente en el desarrollo del control PID de posición del motor DC se utilizara la función de entrada escalón, sin embargo existen las entradas rampa y parabólica entre otras.

1.1.2 Función de entrada escalón.

La entrada función escalón representa un cambio instantáneo en la entrada de referencia. Para el control de posición del motor por ejemplo, si la entrada es una posición angular, una entrada escalón representa una rotación súbita del eje hasta la posición referenciada. Y en el caso de ser una velocidad un cambio desde cero (detenido) hasta la velocidad deseada.

La representación matemática de una función escalón de magnitud R es descrita mediante la ecuación 1.3:

$$r(t) = \begin{cases} R & t \geq 0 \\ 0 & t < 0 \end{cases} \qquad \textit{Ecuación 1.3}$$

Donde R es una constante real. O bien mediante la Ecuación 1.4.

$$r(t) = Ru_s(t) \qquad \textit{Ecuación 1.4}$$

Donde $u_s(t)$ es la función escalón unitario. La función escalón como función del tiempo se muestra en la figura 1.1 y tiene utilidad como señal de prueba ya que su cambio instantáneo de amplitud proporciona información de que tan rápido responde el sistema a entradas abruptas.

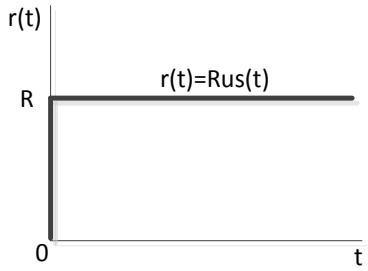


Figura 1. 1 Función escalón unitario. [1.1]

1.1.3 Error en estado estable.

Cuando se asigna una entrada a un sistema de control se dice que se ha dado una referencia para que la salida de este alcance ese valor, sin embargo cuando esto no sucede existe una diferencia entre la referencia y la salida cuando la respuesta transitoria se ha extinguido y esta diferencia es llamada error en estado estable.

Por lo tanto en un sistema de control se busca que el error en estado estable sea mínimo o que este entre un rango de valores aceptables manteniendo todos los requerimientos del sistema dentro de los limites.
En general, el error se puede ver como una señal que debe ser reducida rápidamente a cero, si esto es posible. Con referencia al sistema en lazo cerrado de la figura 1.2, en donde r(t) es la entrada, y(t) es la salida, u(t) es la señal actuante y b(t) es la señal de realimentación. El error del sistema se puede define en la ecuación 1.5:

$$e(t) = señal\ \ de\ referencia - y(t) \qquad \textit{Ecuación 1.5}$$

En donde señal de referencia es la señal que la salida y(t) está siguiendo. Cuando el sistema tiene realimentación unitaria (H(s)=1), la entrada r (t) es la señal de referencia, y el error se representa por la ecuación 1.6.

$$e(t) = r(t) - y(t) \qquad \textit{Ecuación 1.6}$$

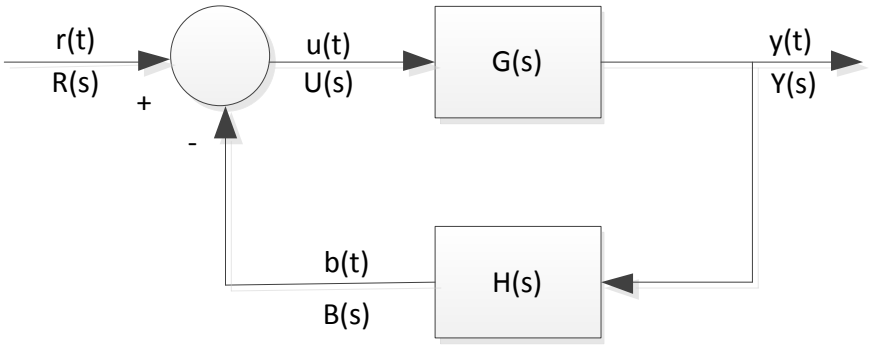


Figura 1. 2. Variables en un diagrama de bloques. [1.1]

Y el error en estado estable se define en la ecuación 1.7:

$$e_{ss} = \lim_{t \to \infty} e(t) \qquad \textit{Ecuación 1.7}$$

En la figura 1.2, dependiendo de la forma de H(s) especialmente si no es unitaria, la señal actuante u (t) puede o no ser el error.

1.1.4 Respuesta al escalón unitario y especificaciones en el tiempo.

La respuesta transitoria y de estado estable puede ser obtenida mediante la aplicación de la señal escalón unitario ver figura 1.3 subsecuentemente se definen los parámetros importantes de esta.

1. Sobrepaso máximo (Mp): Si $y(t)$ es la respuesta al escalón unitario. También es el valor máximo de $y(t)$ y y_{ss} es el valor en estado estable de $y(t)$ y $y_{max} > y_{ss}$. El sobrepaso máximo de $y(t)$ se define en la ecuación 1.8 como:

$$\text{Sobrepaso máximo} = y_{max} - y_{ss} \qquad \textit{Ecuación 1.8}$$

Y en la ecuación 1.9 el sobrepaso máximo se representa como un porcentaje del valor final de la respuesta escalón:

$$\text{Porcentaje de Sobrepaso} = \frac{sobrepaso\ maximo}{y_{ss}} * 100\% \qquad \textit{Ecuación 1.9}$$

2. El sobrepaso máximo es un parámetro a limitar pues se relaciona con la estabilidad del sistema, un sistema con un sobrepaso alto es indeseable, con fines de diseño el sobrepaso suele darse como una especificación en el dominio del tiempo.

3. Tiempo de retardo: el tiempo de retardo t_d se define como el tiempo requerido para que la respuesta escalón alcance el 50% de su valor final.

4. Tiempo de levantamiento: el tiempo de levantamiento t_r se define como el tiempo requerido para que la respuesta al escalón avance del 10% al 90 %de su valor final. Una medida alternativa es representar el tiempo de levantamiento como reciproco de la pendiente de la respuesta al escalón en el instante que la respuesta es igual al 50% de su valor final.

5. Tiempo de asentamiento: el tiempo de asentamiento t_s es el tiempo requerido para que la respuesta al escalón disminuya y permanezca dentro de un porcentaje especifico de su valor final. Una cifra de uso frecuente es 5%.

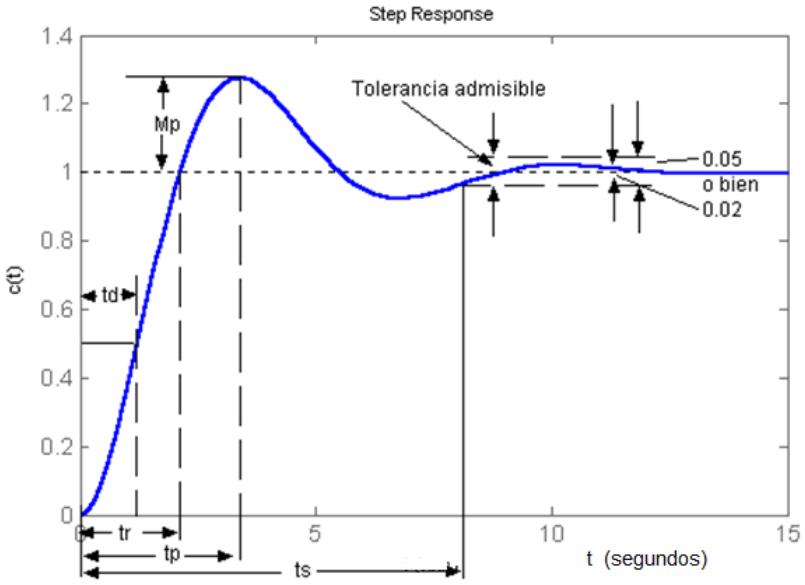


*Figura 1. 3*Función escalón unitario con sus diferentes parámetros. [1.1]

1.2 Acciones de los controladores.

Las acciones de los controladores son las decisiones que toma el controlador para reducir el error entre una referencia y su salida y que son transmitidas a un elemento de control final para que sean ejecutadas, entre los controladores más comunes se tiene el control de dos posiciones, control proporcional, proporcional integral, proporcional derivativo y proporcional integral derivativo PID.

1.2.1 Acción de dos posiciones o de encendido y apagado (On/Off).

En un sistema de dos posiciones el elemento encargado del control final generalmente tiene solo dos estados: encendido y apagado. Así que si se tiene un error la operación de corrección consiste en mantener a un máximo o un mínimo la señal actuante que se denomina m(t) ver ecuación 1.10.

$$m(t) = K1, e(t) > 0$$
$$m(t) = k2, e(t) < 0 \qquad \textit{Ecuación 1.10}$$

Donde k1 y k2 son constantes. Los controladores de 2 posiciones utilizan dispositivos eléctricos para realiza la tarea de control como relés, solenoides etc.

Se puede presentar la operación de un control ON, OFF mediante la figura 1.4, en la parte superior se muestra una onda seno como entrada por lo que la salida toma valores de uno o cero para intentar corregir el error, este es un sistema bastante robusto y es usual encontrarlo en aparatos

eléctricos como hornos o refrigeradores, de modo que si se baja o sube de cierta temperatura se enciende o apaga el actuador según condiciones deseadas.

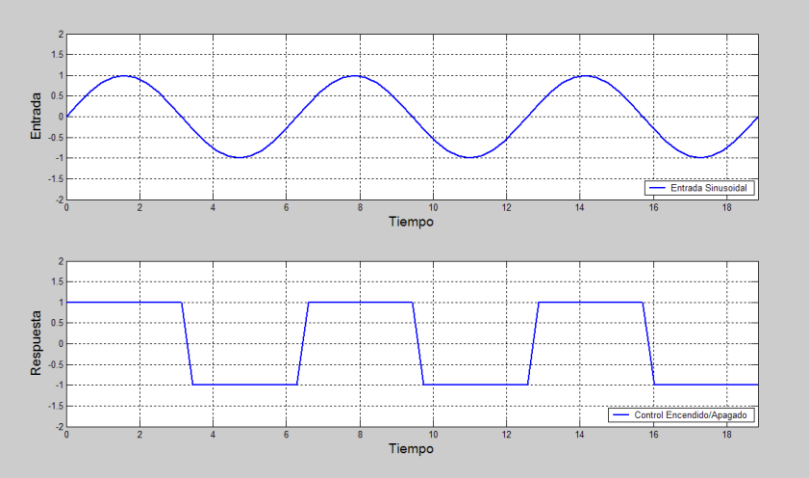


Figura 1. 4. Controlador de dos posiciones. [1.2]

1.2.2 Acción de control proporcional, P.

Para una acción de control proporcional, la relación entre la salida del controlador, *m(t)* y la señal de error, *e(t)* se presenta en la ecuación 1.11:

$$m(t) = Kp * e(t) \qquad \textit{Ecuación 1.11}$$

El controlador proporcional es en general un amplificador del error con ganancia ajustable.

El control proporcional es representado en la figura 1.5 en la cual se aprecia que si el error es de 1.0 la salida del controlador es de 2.0, lo que indica que la ganancia proporcional es de 2, también se aprecia la variable de proceso la cual tiene a estabilizarse luego de un tiempo.

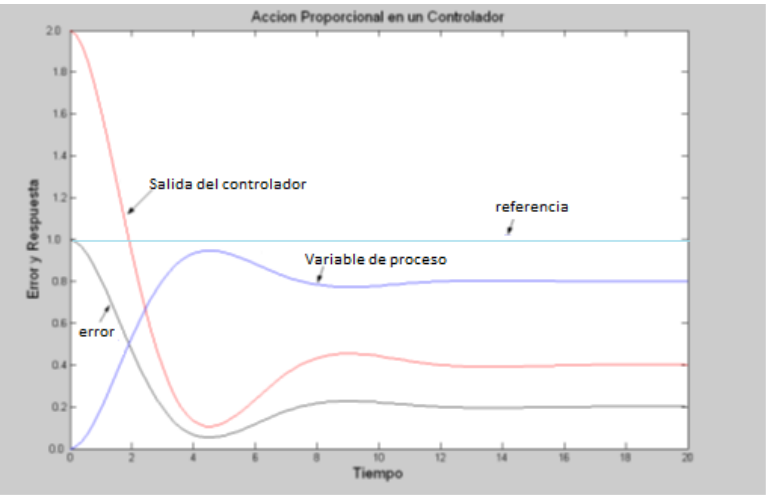


Figura 1. 5 Acción Proporcional en un Controlador [1.2]

Generalmente en la respuesta del control proporcional hay un error en estado estable o desplazamiento (*offset*) para una entrada escalón. Este desplazamiento se reduce a cero idealmente si se incluye la acción de control integral.

1.2.3 Acción de control integral, I.

En una acción de control integral, la rapidez de cambio en la respuesta del controlador, m(t) es proporcional al error, e(t), es decir:

$$\frac{dm(t)}{dt} = Kp * e(t) \qquad \textit{Ecuación 1.12}$$

Por lo que integrando se tiene la ecuación 1.13

$$m(t) = \frac{Kp}{ti}\int_0^t e(t)dt \qquad \textit{Ecuación 1.13}$$

En la figura 1.6 se muestra el resultado en la variable de proceso cuando se utiliza un controlador integral. Si los parámetros del controlador son adecuados es posible disminuir el error en estado estable hasta cero (teóricamente), para llegar a la reducción del error el control integral muestra una respuesta oscilatoria creciente o decreciente y esto se considera un inconveniente. La salida del controlador es el área bajo la curva del error en cualquier instante (integral del error).

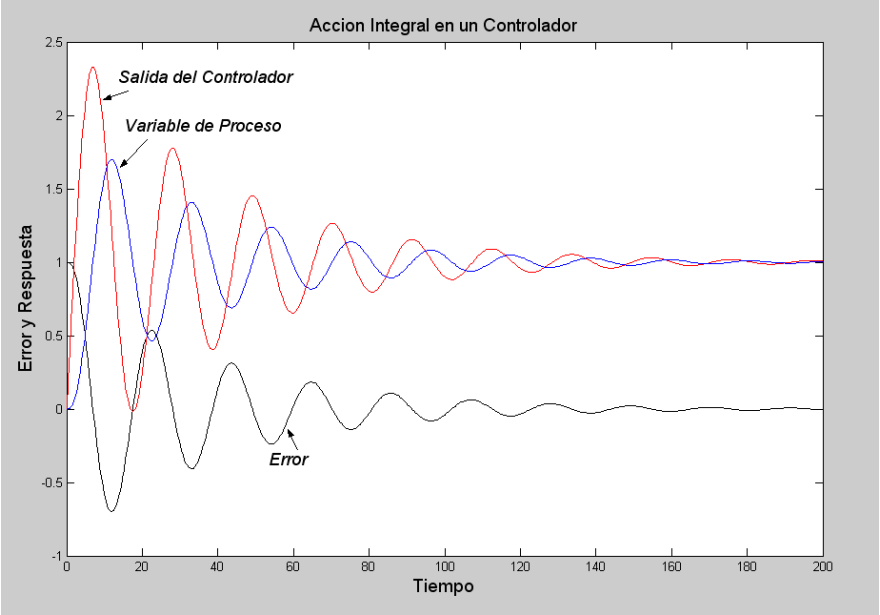


Figura 1. 6 Acción Integral en un Controlador [1.2]

1.2.4 Acción de control proporcional – integral, PI.

La acción de control proporcional – integral, PI, se define mediante la ecuación 1.14,

$$m(t) = Kp * e(t) + \frac{Kp}{ti}\int_0^t e(t)dt \qquad \textit{Ecuación 1.14}$$

Siendo Kp la ganancia proporcional y "ti" el denominado *tiempo integral*. Tanto Kp como "ti" son ajustables.

Significado del tiempo integral

El tiempo integral ajusta la velocidad de la acción de control integral, mientras que un cambio en el valor de Kp afecta las partes tanto proporcional como integral. La Figura 1.7 muestra los perfiles de las acciones proporcional y proporcional-integral de un controlador ante una entrada escalón unitario.

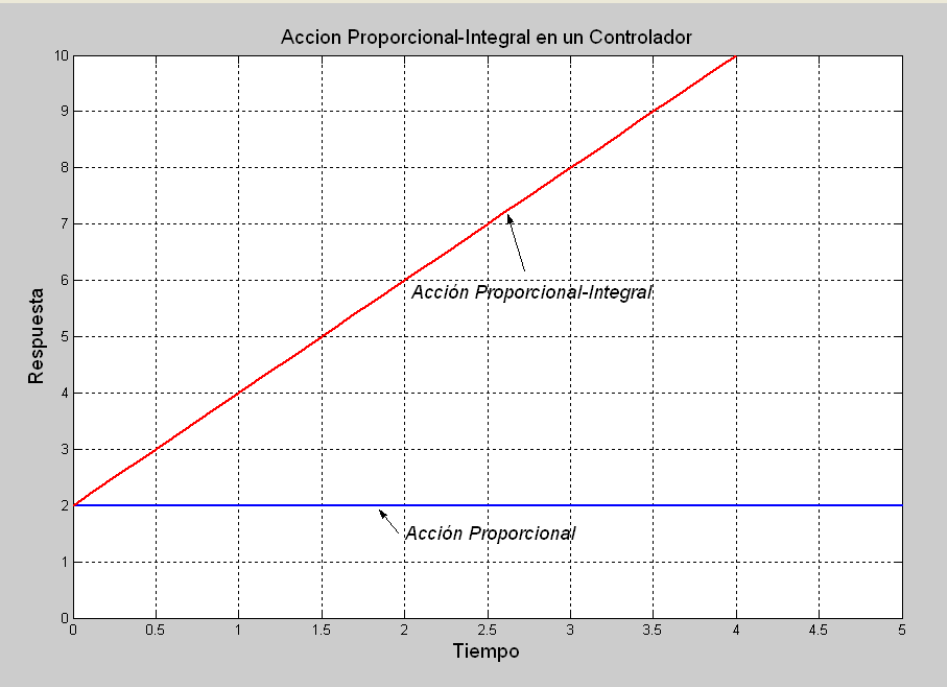


Figura 1. 7 Acciones Proporcional y Proporcional – Integral. [1.2]

Se deduce que la acción proporcional hace una amplificación constante del error alimentado de acuerdo a su ganancia. Para el controlador proporcional integral, la respuesta inicial es igual a la ganancia proporcional y esta respuesta se repite sumada para períodos de tiempo igual al tiempo integral.

1.2.5 Acción de control proporcional – derivativa, PD.

La acción de control proporcional – derivativa, PD, se define mediante la ecuación 1.15:

$$m(t) = Kp * e(t) + Kp * \ td\frac{de(t)}{dt} \qquad \textit{Ecuación 1.15}$$

Siendo Kp la ganancia proporcional y td una constante llamada "*tiempo derivativo*". Ambos parámetros son ajustables.

El control derivativo siempre se usa junto con la acción de control proporcional, con la acción PD se obtiene un controlador que responde a la velocidad de cambio del error y con esto proporciona una corrección significativa antes que la magnitud del error sea demasiado grande. Este controlador permite utilizar ganancias proporcionales altas y con esto mejorar la precisión y disminuir las oscilaciones del sistema cuando se utiliza un controlador PID.

Significado del tiempo derivativo.

El tiempo derivativo es el intervalo de tiempo durante el cual la acción de la velocidad hace avanzar el efecto de la acción de control proporcional.

Si la señal de error es una función rampa unitaria, la salida del controlador se convierte en la que se muestra en la Figura 1.8. La acción de control derivativa tiene un carácter de previsión.

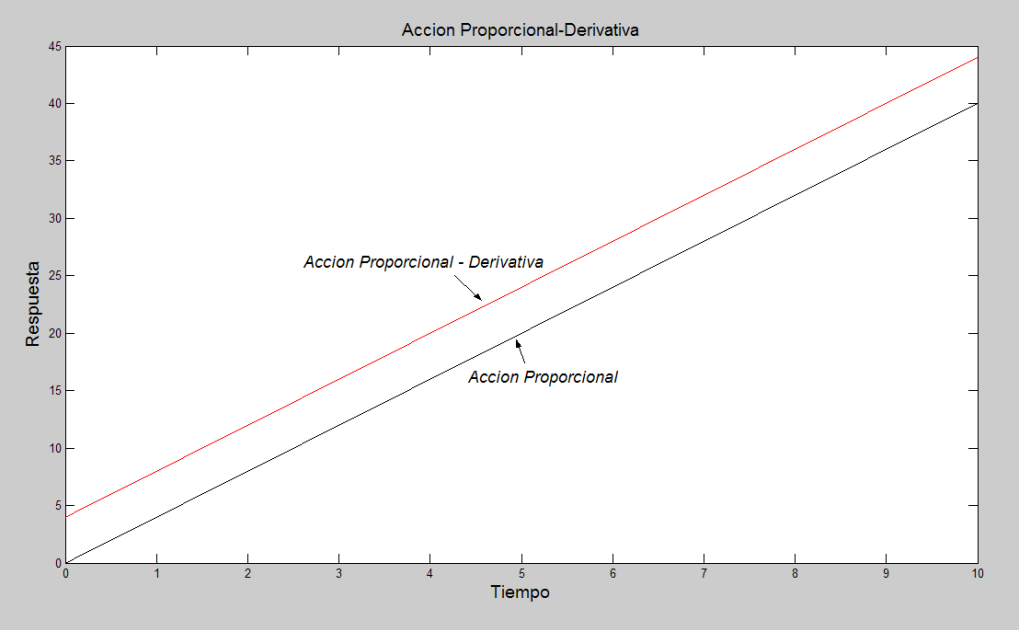


Figura 1. 8 Acción Proporcional y Proporcional – Derivativa [1.2]

La Figura 1.9 muestra las respuestas de los controladores proporcional, proporcional-integral y proporcional derivativo para el proceso utilizado en los casos anteriores.

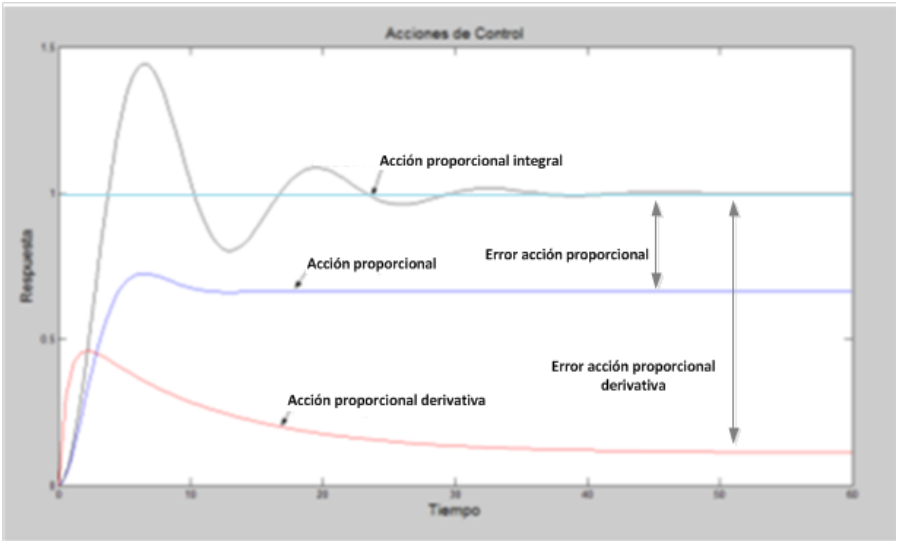


*Figura 1. 9*Acciones de control P, PI y PD [1.2].

1.2.6 Acción de control proporcional – integral – derivativa, PID.

La combinación de las acciones de control proporcional, integral y derivativo dan lugar al controlador PID ver ecuación 1.16, el cual cuenta con las características de cada una de estas acciones de control individuales.

$$u(t) = K_p\left(e(t) + \frac{1}{t_i}\int_0^t e(t)dt + t_d\frac{de(t)}{dt}\right) = P + I + D \qquad \textit{Ecuación 1.16}$$

En la Figura 1.10 se muestra la respuesta del control proporcional-integral-derivativo a una variación escalón unitario en su variable de entrada para el proceso estudiado en los casos anteriores.

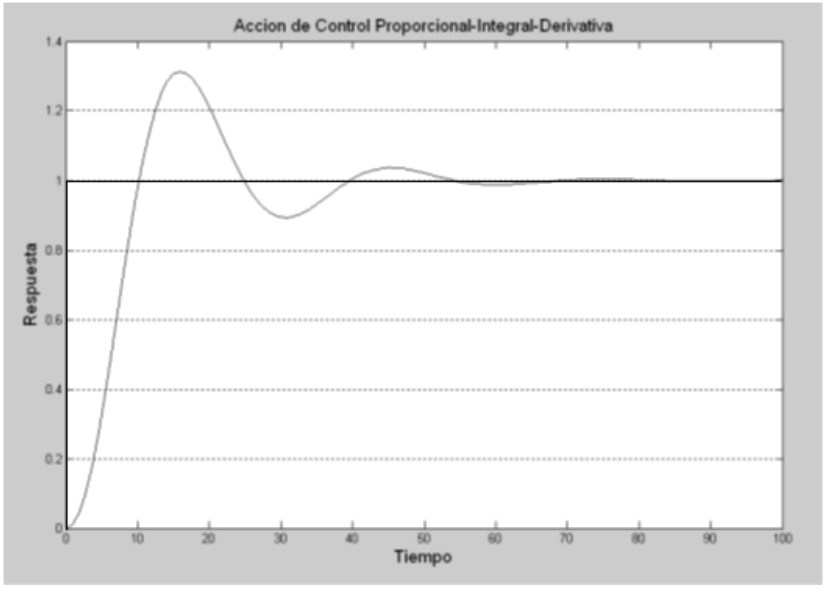


Figura 1. 10 Acción de control proporcional – integral – derivativo [1.2].

1.3 Configuración del controlador.

La compensación del controlador se refiere al lugar donde el controlador está colocado respecto al proceso que se quiere controlar.

Compensación serie: como su nombre lo indica esta configuración se realiza colocando en serie el controlador y el proceso que se quiere controlar (planta). Se ilustra en la figura 1.11.

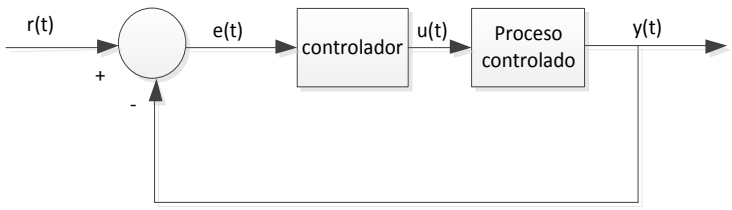


Figura 1. 11 Configuración compensación serie.

Por simplicidad se utilizara esta configuración para la implementación del PID. Se dice que el esquema antes expuesto tiene un grado de libertad ya que solo hay un controlador en el sistema, aun cuando el controlador pueda tener más de un parámetro que pueda variar.

El controlador a poner en serie con la planta es el PID, este tiene algunas ventajas significativas que lo han hecho un estándar en diversos procesos de la industria, aquí se citan algunas de esas ventajas:

- El controlador PID al utilizar la realimentación provee de estabilidad al sistema ante cambios externos a este que llegan a repercutir en su control, por ejemplo condiciones ambientales o deterioro de los componentes del elemento a controlar.

- El término derivativo proporciona una acción anticipativa sobre la respuesta al sistema.

- El término integral reduce el error en régimen permanente.

- Existen sencillas reglas heurísticas que permiten obtener los parámetros del controlador PID. Dichas reglas hacen posible el ajuste del controlador, sin presuponer un gran conocimiento en teoría de control automático.

- El controlador PID se puede adquirir como un módulo compacto, donde los distintos parámetros del controlador se pueden ajustar manualmente. Actualmente muchos de los PID industriales proporcionan opciones de auto sintonía.

La siguiente tabla resumen proporciona una idea de los efectos de las ganancias en el controlador PID.

Respuesta en lazo cerrado	Tiempo de subida	Sobrepaso	Tiempo de asentamiento	Error en estado estacionario	Estabilidad
Incrementando Kp	Disminuye	Aumenta	Pequeño aumento	Disminuye	Degrada
Incrementando Ki	Disminuye un poco	Aumenta	Pequeño aumento	Disminuye	Degrada
Incrementado kd	Disminuye un poco	Disminuye	Disminuye	Cambio menor	Mejora

Tabla 1.1 Efectos que ocurren en el controlador PID al incrementar las constantes.

1.4 Control de posición y velocidad por medio de un controlador PID para un motor de corriente directa (dc).

Para el control del motor dc el primer paso es identificar la variable a controlar y la variable sobre la cual se ejerce control. Para el caso las variables a controlar serán la velocidad y la posición del motor y la variable de actuación el ciclo de trabajo del PWM (ver 1.7.1).

Calcular el error en el caso de la posición del eje, consiste en calcular la diferencia entre la posición, de referencia que se tendrá en la entrada r(t) y obtener la posición a lo largo del tiempo a través del encoder incremental el cual proporciona realimentación al sistema para llevar por medio del controlador y la señal actuante PWM la reducción del error a cero.

Se define este error para el caso de posición como:

$$ep(t) = P_{ref} - P \qquad \textit{Ecuación 1.17}$$

Donde Pref es la referencia y P la posición decodificada proporcionada por el encoder, el error de velocidad se deduce de forma análoga, por lo que:

$$ev(t) = v_{ref} - v \qquad \textit{Ecuación 1.18}$$

Sin embargo las descripciones anteriores del error y del controlador no están pensadas para un sistema digital por lo que para hacer uso de la FPGA y los datos de posición y velocidad que proporciona el encoder, la ecuación PID mostrada se debe discretizar**,** en el caso más básico la ecuación PID se puede hacer mediante un método rectangular como sigue.

- La integral se substituye por definición como la suma de los valores anteriores del error, además se limita en el tiempo (sumar los últimos k errores). Originalmente como se ha descrito se debería agregar un tiempo de integración sin embargo este es constante y puede ser absorbido en la constante ki.

- La derivada de una función puede ser calculada como la diferencia entre 2 puntos que para el caso son los errores sucesivos en un tiempo determinado (tiempo derivativo), sin embargo este tiempo se asume implícito en la constante kd.

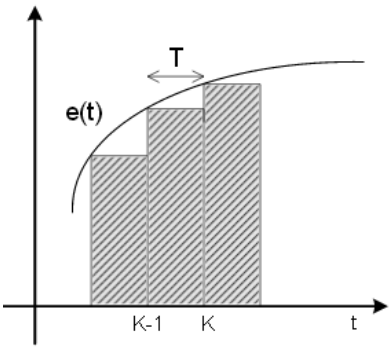


Figura 1. 12 suma de los K errores, [1.3].

Apoyándose en la figura 1.12, el término proporcional del PID para tiempo discreto se convierte en:

$$K_p e(t) = K_p e(n)$$ *Ecuación 1.19*

El término derivativo:

$$td\,\frac{\partial e(t)}{\partial t} = td\,\frac{e_k - e_{k-1}}{T}$$ *Ecuación 1.20*

Y el término integral en:

$$\left(\frac{1}{ti}\right)\int_0^t e(t)\,dt = \left(\frac{1}{ti}\right)\sum_{i=0}^{k-1} T e_i$$ *Ecuación 1.21*

Por lo que la expresión PID para tiempo discreto queda de la forma:

$$u_k = kp\,\{e_k + \frac{T}{T_i}\sum_{i=0}^{k-1} e_i + \frac{Td}{T}(e_k - e_{k-1})\,\}$$ *Ecuación 1.22*

Por otra parte, la variable de control en el instante de tiempo *uk* se puede calcular sobre la base de su valor en el instante de tiempo anterior u (k-1):

El valor en un tiempo anterior u (k-1) es:

$$u_{k-1} = Kp\,\{e_{k-1} + Ti\sum_{i=0}^{k-2} e_i + Td(e_{k-1} - e_{k-2})\}$$ *Ecuación 1.23*

Restando la expresión u (k-1) (ec1.23) de la de u (k) (ec1.22), se obtiene:

$$u(t_k) = u(t_{k-1}) + q0e_k + q1e_{k-1} + q2e_{k-2}$$ *Ecuación 1.24*

Dónde:

$$q0 = Kp\left(1 + \frac{Td}{T}\right)$$ *Ecuación 1.24.1*

$$q1 = Kp\left(-1 - 2\frac{Td}{T} + \frac{T}{Ti}\right)$$ *Ecuación 1.24.2*

$$q2 = Kp\left(\frac{Td}{T}\right)$$ *Ecuación 1.24.3*

La ecuación 1.24 es un algoritmo de control que se denomina algoritmo de velocidad del PID, mientras que el algoritmo de la ecuación 1.22 se denomina algoritmo de posición del PID.

1.5 Sintonización del PID.

Sintonizar un controlador PID significa establecer el valor que deben tener los parámetros de ganancia proporcional, ganancia integral y ganancia derivativa, para que el sistema responda en una forma adecuada según se requiere.

Sintonizando el controlador se obtienen las características de sobrepaso, tiempo de asentamiento y error de estado estable que mejor convengan al proceso. Para sintonizar el PID existen métodos de lazo cerrado y métodos de lazo abierto además de la sintonización por métodos heurísticos.

Métodos en Lazo Cerrado: la información de las características del lazo se obtienen a partir de una prueba realizada en lazo cerrado, usualmente con un controlador con acción proporcional pura ver figura 1.13.

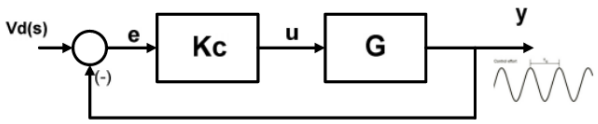


Figura 1. 13 Respuesta de procesos en lazo cerrado [1.4].

Métodos en Lazo Abierto: las características estáticas y dinámicas de la planta (Elemento Final de Control +Proceso + Transmisor) se obtienen de un ensayo en lazo abierto, generalmente la respuesta a un escalón, ver figura 1.14.

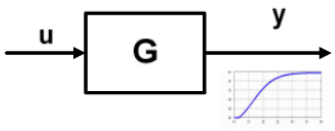


Figura 1. 14 Respuesta en procesos a lazo abierto [1.4].

Métodos heurísticos: Este método es comúnmente utilizado en controladores digitales ya que el cambio de las constantes implica cambios en software de programación y no de hardware, se utiliza generalmente cuando no se dispone información suficiente acerca de la planta que se desea controlar, otra ventaja es que no se requieren grandes conocimientos en la teoría de control. En otras palabras este es un método de prueba y error, sin embargo fundamenta su procedimiento conociendo el comportamiento que causa la modificación de la constante proporcional, integral y derivativa como se muestra en la tabla 1.1.

Para comprobar el funcionamiento del sistema de control de posición y velocidad del motor, se utiliza una función de entrada escalón cuyo máximo es la referencia seleccionada de esta forma se prueba el funcionamiento de la planta ante un cambio repentino en la entrada.

Periodo de Muestreo

Como periodo de muestreo se toma normalmente 1/10 del tiempo de subida del sistema en lazo cerrado. El tiempo de subida se define como el tiempo necesario para que el sistema pase del 10% al 90% del valor final de la respuesta.

Otro criterio que se puede usar es 1/10 de la constante de tiempo del sistema. La constante de tiempo de un sistema se define como el tiempo necesario para que la respuesta a un escalón unitario alcance el 63% del valor final de la respuesta. Para un sistema que logra su valor final a los 2 seg. Se toma como periodo de muestreo T igual a 0.12 seg que es 1/10 de la constante de tiempo del sistema.

1.6 Efecto WINDUP.

La saturación del accionador es un fenómeno que debe ser corregido ya que si el controlador satura el accionador (el accionador trabaja en su máximo valor) es posible que la acción integral tarde tiempo para cambiar de signo y ejercer una acción de signo contrario por lo que se seguirá enviando una señal positiva (debida a un error positivo, creciente) a pesar que las acciones proporcional y diferencial actúen de forma correcta enviando una señal negativa (debida a un error negativo, decreciente). Subsecuentemente el accionador mantendrá al máximo el control sobre el sistema y el sistema cae en un estado de inestable.

Para entender mejor este problema se presenta un controlador PI que presenta saturación ver figura 1.15, ante un error apreciable el controlador P buscara corregir este error por medio de su acción proporcional *u1(t),* mientras esto sucede la acción integral no presenta respuesta aun pero comienza a integrar el error pudiendo suceder que la respuesta integra *u2(t)* adquiera una magnitud que impida transitoriamente que el controlador PI salga de la región de saturación a pesar que el error haya disminuido.

Entonces para reducir la acción integral *u2(t)* es necesario que el error cambie de signo, la disminución de *u2(t) es lenta y hace que la acción de control PI u(t)* mantenga su signo a pesar que el error tiene sentido contrario.

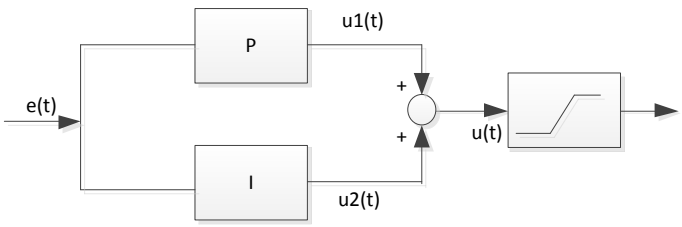


Figura 1. 15 Controlador PI con saturación [1.5]

1.6.1 Algoritmos anti-WINDUP.

Ya que el integrador *u2(t)* aumenta su salida para reducir el error aun cuando la señal de control u(t) este saturada. Esto hace que cuando el sistema se acerca al punto de referencia el término integral haya crecido demasiado (efecto Windup) y se sobrepase el valor deseado. Ya que la única forma de reducir el valor del término integral es mediante un cambio de signo a su entrada (señal de error).

Hay diferentes técnicas para evitar este fenómeno denominadas “algoritmos anti-Reset windup (ARW)”, todas ellas de fácil implementación digital:

- Limitar el término integral en un valor determinado.
- Cortar la acción integral durante la saturación.
- Limitar el termino PID.

En el diseño del controlador PID de posición/velocidad del motor dc se optará por cortar la acción integral durante la saturación además de limitar el termino PID.

1.7 Control del motor, el puente H.

Para invertir el giro de un motor DC basta con cambiar la polaridad en sus terminales, esta tarea es comúnmente realizada por un circuito llamado “puente H” debido a su similitud con una H mayúscula. Un circuito general y de fácil comprensión es mostrado en la figura 1.16.

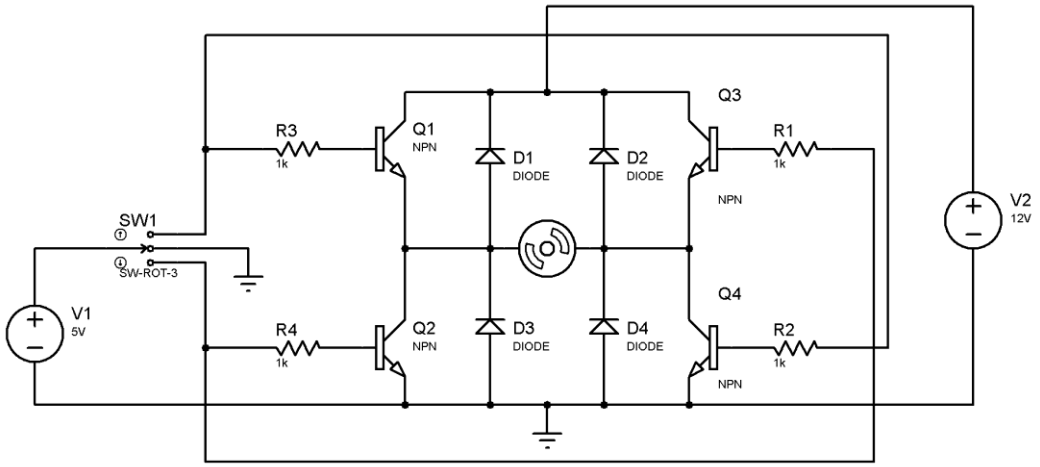
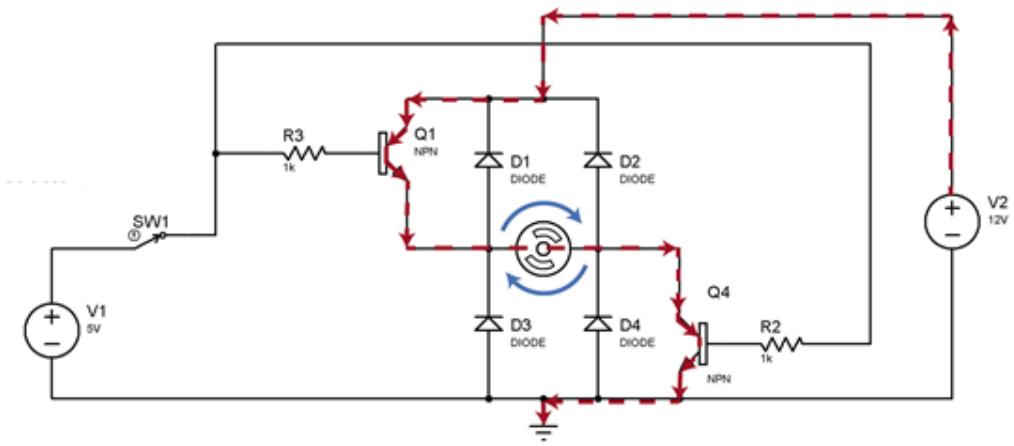


Figura 1. 16 Puente H para motor de corriente continua. [1.7]

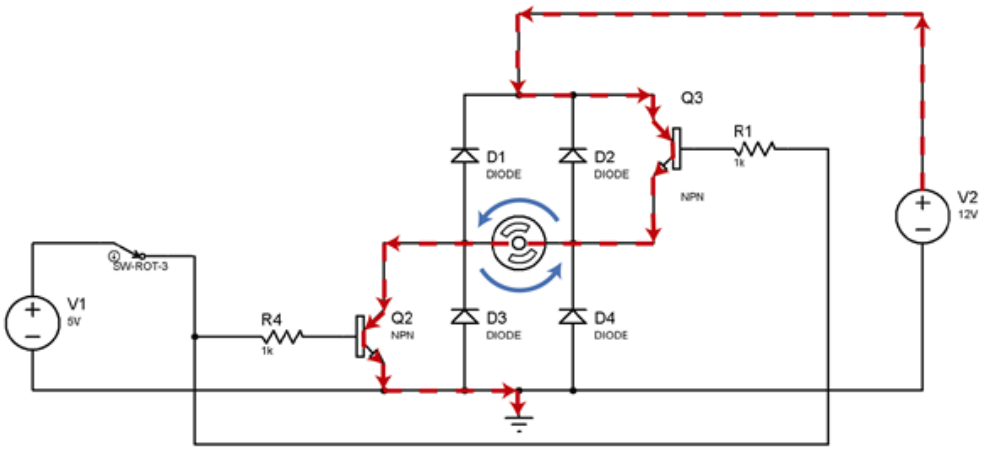
Funcionamiento:

En un inicio sino se acciona SW1, los transistores no están activados, la figura 1.17 muestra que sucede cuando SW1 se acciona hacia una de sus 2 posiciones.

En la posición superior el circuito establece continuidad entre los transistores Q1 y Q4 ver figura, lo que hace girar al motor en la dirección horaria, cuando el SW1 se acciona a su otra posición que no es reposo se activan los transistores Q3 y Q2 por lo que el motor ahora tiene la polaridad invertida y girara en sentido anti horario.



a)



b)

Figura 1. 17 a) Conducción de Q1 Y Q4, b) Conducción de Q3 Y Q2. [1.7]

Sin embargo si se decide usar un motor que tenga un mayor consumo de corriente se debe aumentar la capacidad de los semiconductores a usar, algunos fabricantes de motores proporcionan los drivers para estos mismos.

Para el motor DC que se desea controlar se utilizara el circuito integrado L298N que tiene como ventaja paralelar sus entradas y salidas con el fin de incrementar su capacidad eléctrica para el manejo de motores de hasta 4 Amperios. El valor de Rs en la figura depende si se desea medir la cantidad de corriente que están utilizando los motores que se controlan sin embargo si esto no es de interés Rs=0.

Esta configuración se muestra en la figura 1.18a.

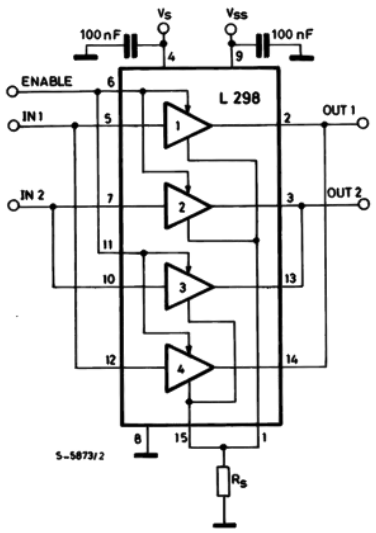


*Figura 1. 18*a paralelo del L298N.

Las entradas IN1 e IN2 sirven para controlar el giro del motor según la tabla de verdad mostrada en la figura 1.20b donde C= IN1 Y D= IN2

Entradas		Función
Ven H	C=H; D=L	Adelante
	C=L ; D=H	Marcha atrás
	C=D	Paro rápido
Ven= L	C=X; D=X	Sin giro
L= bajo "0"	H=alto "1"	X=Don't Care

*Figura 1. 18*b tabla de verdad del L298.

El PWM se debe aplicar en la entrada ENEABLE (Ven) para un mejor funcionamiento del paro del motor ya que este es controlado por las entradas IN1 e IN2 con otras señales de control.

Ya que los dispositivos FPGA suelen dar corrientes muy bajas en sus pines alrededor de 16mA (a un voltaje LVTTL 0v-3.3V), poner directamente un pin de salida en una entrada IN1, IN2 del L298N podría dañar este Pin del FPGA e incluso dañar la tarjeta. La solución para este y muchos circuitos de control es usar un circuito optoacoplador que aísle el circuito de control con el circuito de potencia. La tierra del led del optoacoplador debe estar conectado a la tierra de la tarjeta FPGA y esta no debe ser unida a la tierra del circuito de potencia ver figura 1.19, además de esto se debe tener en cuenta que el optoacoplador es colector abierto a su salida, por lo tanto para tener una señal output=1 en la figura 1.19 se debe poner a cero la entrada input.

También es importante usar siempre los diodos expuestos en la figura 1.16 con el fin de limitar la corriente que circula por el motor a su fuente de poder, el valor usado para R1 es 220 Ohm y para R2 es 1kOhm.

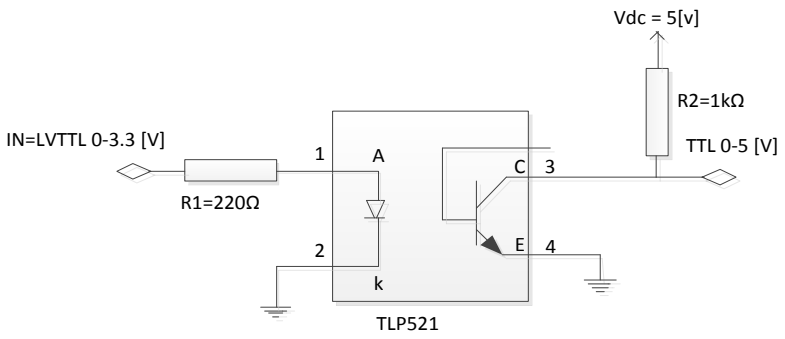


Figura 1. 19 Circuito optoacoplador.

1.7.1 Modulación por ancho de pulso (PWM).

La modulación por ancho de pulsos, por sus siglas Pulse Widht Modulation (PWM) es una técnica de con la cual se produce el efecto de una señal analógica constante a partir de la variación del ciclo de trabajo de una señal digital que es generalmente una forma de onda cuadrada que varía entre 0 y 1. El ciclo de trabajo es el tiempo en el cual la señal está en un estado lógico alto como un porcentaje del tiempo total que esta toma para completar un ciclo completo ver figura 1.20.

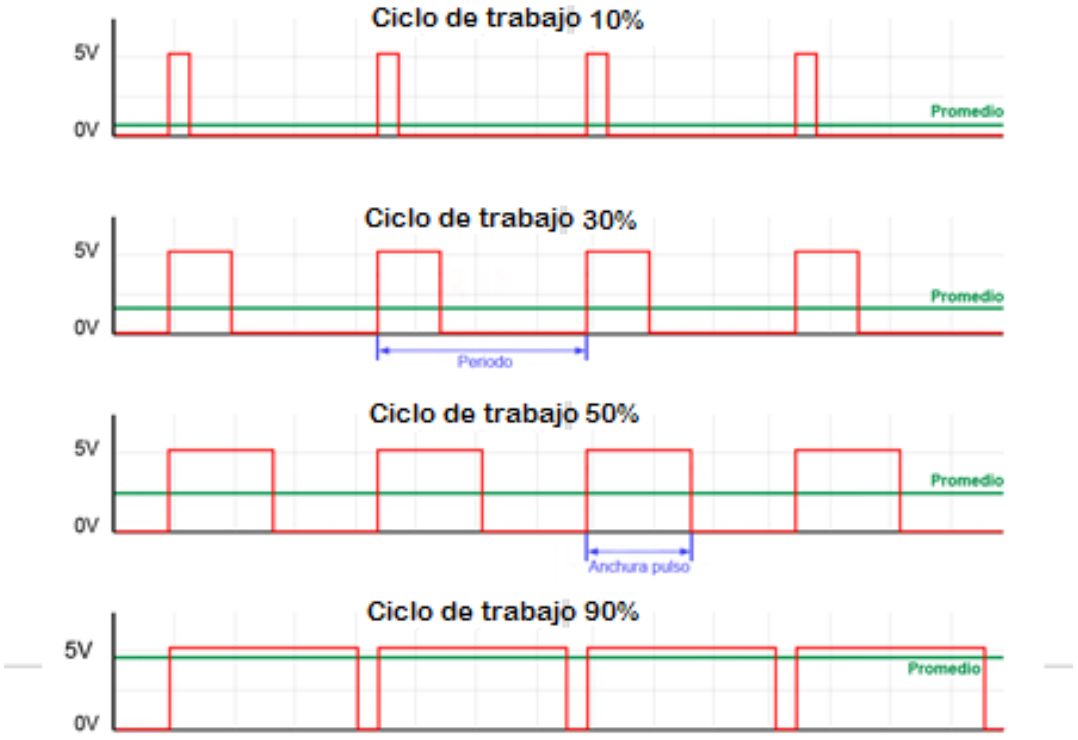


Figura 1. 20. Descripción de PWM. [1.8]

El ciclo de trabajo del PWM es lo que afecta directamente al motor pues determina la cantidad de energía que le es enviada ver figura 1.20. El ciclo de trabajo está relacionado directamente con el tiempo de encendido ver figura 1.21 y el periodo de la señal.

En la mayoría de controles electrónicos que usan PWM la frecuencia de trabajo se mantiene fija mientras lo que se varía es el ciclo de trabajo.

Para el control de motores DC con PWM se deben tener en cuenta ciertas características inherentes a este, por ejemplo la frecuencia de operación del PWM tiene relación directa con una pérdida de potencia, esto se debe a que una pérdida de potencia ocurre cada vez que ocurre una transición de 0 a 1 o viceversa.

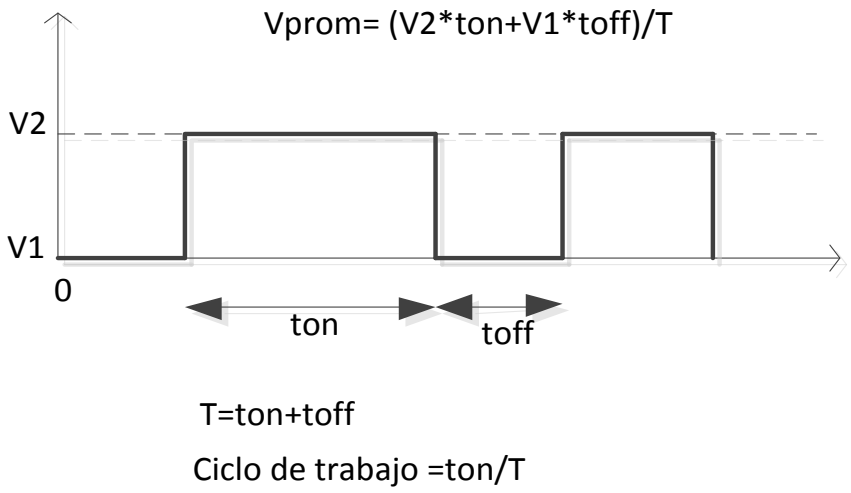


Figura 1. 21 Representación del ciclo de trabajo de una señal cuadrada.

Para utilizar el PWM en el puente H se tiene la opción de introducir la señal directamente a las entradas IN1, IN2 en la figura 1.18a o aplicar la señal al enable del L298N, ya que se utilizaran las entradas IN1 e IN2 como señales para el control de posición del motor dc, se opta por utilizar el PWM en el "enable" y así utilizar la característica de frenado rápido que ofrece el driver L298N.

Ya que la señal de salida del PID debe actuar de forma que aumente o reduzca el ciclo de trabajo de la señal PWM para que de esta forma aumente o disminuya la velocidad del motor dc, se debe utilizar un método que "traduzca" la señal de salida PID a la señal actuante PWM. Para realizar esta tarea se compara el valor de un contador cuya frecuencia de desbordamiento depende de la frecuencia que se desee asignar al PWM y la salida del PID de modo que ambas señales tengan los mismos límites (mínimos y máximos) para realizar una comparación apropiada, por ejemplo si se comparan números de 3 bit del PID con números de 3 bit del contador, en decimal se comparan números de 0 a 7 del PID vs números de 0 a 7 del contador. Si se evalúa la condición "Es el valor de salida del PID <= El valor del contador" y es cierta entonces la señal PWM toma el valor lógico "0" y caso contrario toma el valor lógico "1" de esta forma se cambia el tiempo en alto/bajo de la señal PWM.

Por ejemplo en la figura 1.22 ya que el contador se desborda con una frecuencia existe un periodo (tiempo 1 o tiempo 2) en el cual la comparación del contador con la señal de salida del PID provoca que la señal PWM se mantenga en estado alto "1" más tiempo en dicho periodo siempre y cuando "señal PID >contador".

	Contador	Salida PID	PWM	
	0	7	1	Tiempo 1
	1	7	1	
	2	7	1	
	3	7	1	
	4	7	1	
	5	7	1	
	6	7	1	
Desbordamiento	7	7	0	
Reinicio del contador	0	5	1	Tiempo 2
	1	5	1	
	2	5	1	
	3	5	1	
	4	1	0	
	5	1	0	
	6	1	0	
	7	1	0	

Figura 1. 22 Modificación del ciclo de trabajo del PWM por comparación con un contador[1.9]

1.8 Motores de corriente continua.

Los motores de corriente directa tienen muchas aplicaciones en la industria, debido a la facilidad en su control de posición, velocidad, par, etc. Además de que proporcionan alto par a bajas velocidades por lo que es normal encontrarlos en procesos automatizados que requieran movimiento de ejes, bandas, brazos robóticos entre otros. En el esquema 1.1 se muestra una forma de clasificación de los motores de corriente directa.

Para la aplicación del PID se usa un motor de excitación independiente particularmente un motor DC de imán permanente cuyas características se presentan a continuación.

1.8.1 Motor de excitación independiente.

Esta configuración es tal que la alimentación del rotor y el estator son dos fuentes de tensión independientes. Ya que el estator tiene su fuente independiente la incidencia de la carga del motor sobre su campo es nula teóricamente por lo que el par es prácticamente constante ver figura 1.23.

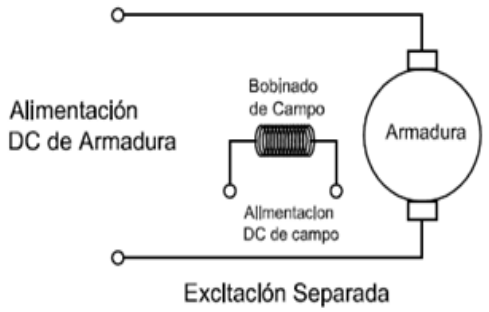
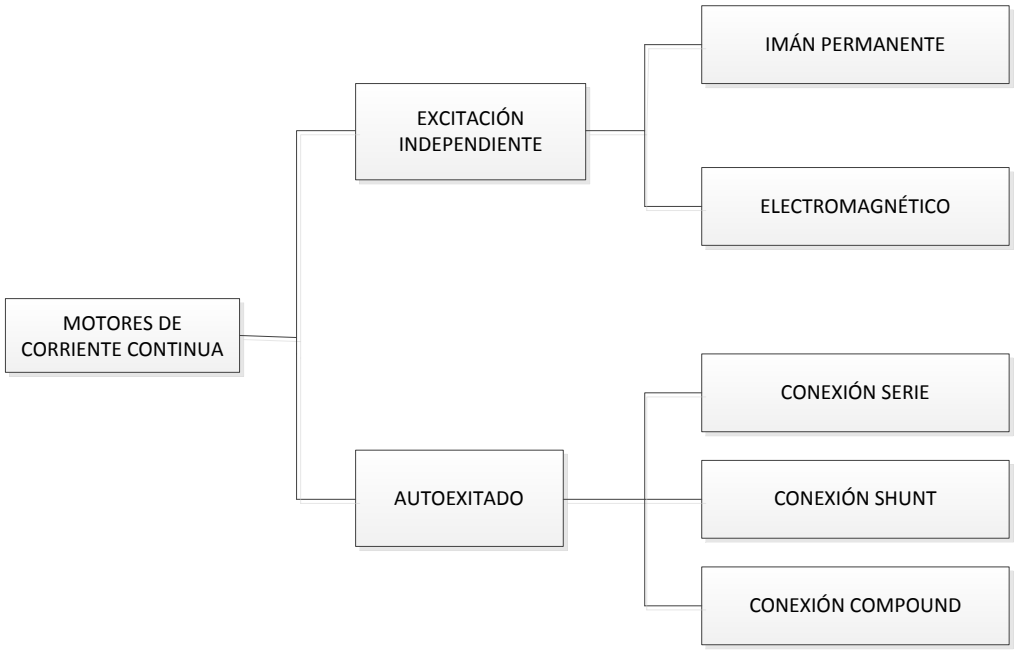


Figura 1. 23 motor de excitación independiente

El motor de excitación independiente es el más adecuado para cualquier tipo de regulación, por la independencia entre el control por el inductor y el control por el inducido.



Esquema 1.1. Clasificación de los motores DC

1.8.2 Motores de imán permanente.

Existe otro tipo de motores que se incluyen en esta categoría llamados motores de imán permanente el cual tiene como fundamento el reemplazo de un bobinado de campo por imanes permanentes, su uso es muy extendido en maquinaria industrial, maquinaria de embalaje, domótica, servomotores entre otros. En la figura 1.24 se muestran los imanes fijos que crean el campo.

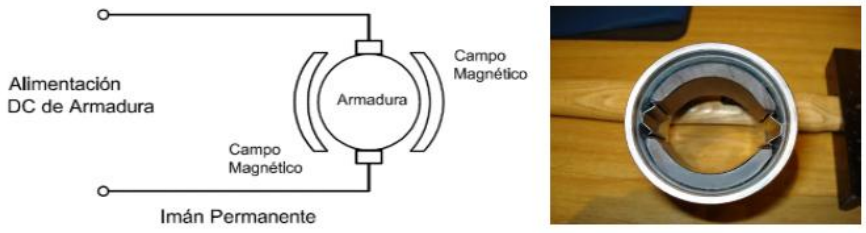


Figura 1. 24. Motor de imán permanente

Los motores de Imán permanente sino son sometidos a rangos de trabajo fuera de sus especificaciones presentan como avería más frecuente el desgaste en as escobillas, estos poseen 2 líneas de conexión en las cuales se suministra la tensión DC, la inversión de giro se realiza solamente con invertir la polaridad en estas líneas, pueden dar grandes velocidades sin embargo no suelen dar el par adecuado para ciertas aplicaciones que requieren de fuerza por lo que generalmente para estas aplicaciones requieren reductores de velocidad para el incremento de su par, sin embargo con el desarrollo de la tecnología de imán permanente es posible alcanzar una relación entre el par , velocidad y volumen a un costo "razonable".

1.9 Codificadores rotatorios.

Los codificadores rotatorios (conocidos genéricamente como encoder) son mecanismos utilizados para entregar la posición, velocidad y aceleración del rotor de un motor, convirtiendo estas magnitudes físicas en un código digital. Sus principales aplicaciones incluyen aplicaciones en robótica, lentes fotográficas, aplicaciones industriales que requieren medición angular, equipos electrónicos que requieran movimiento controlado de ejes, etc.

Los tipos más comunes de Encoder se clasifican en: absolutos y relativos (conocidos también como incrementales). Los encoder absolutos pueden venir codificados en binario o gray ver figura 1.25. Dentro de los encoder incrementales, se encuentran los encoder en cuadratura, ampliamente utilizados en motores de alta velocidad y en aplicaciones en las que interesa conocer la dirección del movimiento del eje además de la velocidad.

El tipo común de encoder incremental consiste de un disco acoplado al eje del motor que contiene un patrón de marcas o ranuras que son codificados por un interruptor óptico (par led/fotodiodo o led/ fototransistor) generando pulsos eléctricos cada vez que el patrón del disco interrumpe y luego permite el paso de luz hacia el interruptor óptico a medida que el disco gira. Esto produce una secuencia que puede ser usada para controlar el ángulo de giro, la dirección del movimiento e incluso la velocidad.

Un parámetro importante de los encoder incrementales son los pulsos por revolución que es un parámetro que determina la resolución del encoder. Desde un encoder incremental no se puede determinar la posición angular absoluta del eje.

Para poder determinar la posición relativa a un punto de referencia (cero), el encoder incremental debe incluir una señal adicional que genera un pulso por revolución, denominada índice.

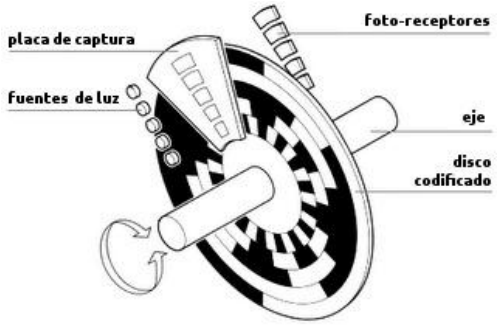


Figura 1. 25. Encoder con disco codificado con código binario [1.10]

1.9.1 Encoder absoluto.

Los **encoder absolutos** emiten un valor numérico codificado único para cada posición del eje. En las tareas de posicionamiento particularmente, los encoder absolutos liberan al controlador de las tareas de cálculo e impiden el aumento de los costes al no necesitarse componentes de entrada adicionales, La figura 1.25 es una muestra de un disco codificado de un encoder absoluto.

Además, ya no es necesario ejecutar el punto de referencia tras una caída de tensión o cuando la máquina se conecta, ya que se proporciona de inmediato el valor de posición actual. Los **encoder absolutos en paralelo** transmiten el valor de posición a la electrónica de análisis a través de varios cables en paralelo. Los **encoder absolutos en serie** transmiten los datos de salida por medio de protocolos e interfaces estandarizadas.

1.9.2 Encoder incremental de cuadratura.

Los encoder incrementales de cuadratura se caracterizan por tener 2 canales de salida, comúnmente llamados "canal A" y "Canal B desfasados 90 grados entre sí, ver figura 1.26b. se puede saber la velocidad de rotación y la posición mediante conteos con un solo canal el cual generalmente es el A, sin embargo es imposible saber cuál es la dirección del movimiento sino se dispone de otro canal para la comparación en este caso el canal B, de este modo el sentido de giro está determinado por el nivel de alguna de las señales de salida con respecto de la segunda, por ejemplo, en el modo 1X (modo normal), si la señal en A durante la transición de bajada (A : 1→0) el nivel lógico de la señal B está en Alto (B=1) implica que está girando en el sentido de las manecillas del reloj (CW), mientras que si B está en la transición de bajada y A=1 implica que está girando en sentido anti horario CCW ver figura 1.7 correspondiente al modo 1X**.**

Una característica importante de los encoder incrementales es el número de pulsos por revolución, abreviado PPR, esta da lugar a la resolución del encoder que es sumamente importante a la hora de tratar con aplicaciones que requieran un posicionamiento preciso del eje. Entre más PPR se tengan se tiene una mejor resolución, de modo que si un encoder proporciona 360PPR entonces se tiene un avance en grados de:

$$Avance = \frac{\#\ 1\ REV\ (Grados)}{PPR} = \frac{360^0}{360PPR} = 1^0/pulso \qquad \text{Ecuación 1.25}$$

Y para un encoder con 1600PPR

$$Avance = \frac{\#\ 1\ REV\ (Grados)}{PPR} = \frac{360^0}{1600PPR} = 0.225^0/pulso \qquad \text{Ecuación 1.26}$$

1.10 Decodificación del encoder.

Las señales de cuadratura pueden ser decodificadas para medir desplazamiento angular y la dirección de rotación como se muestra en la figura 1.27, los pulsos de salida se muestran en columnas separadas como rotación en dirección de las manecillas de reloj (Clock Wise, CW) y rotación en sentido anti horario (Counterclockwise, CCW).

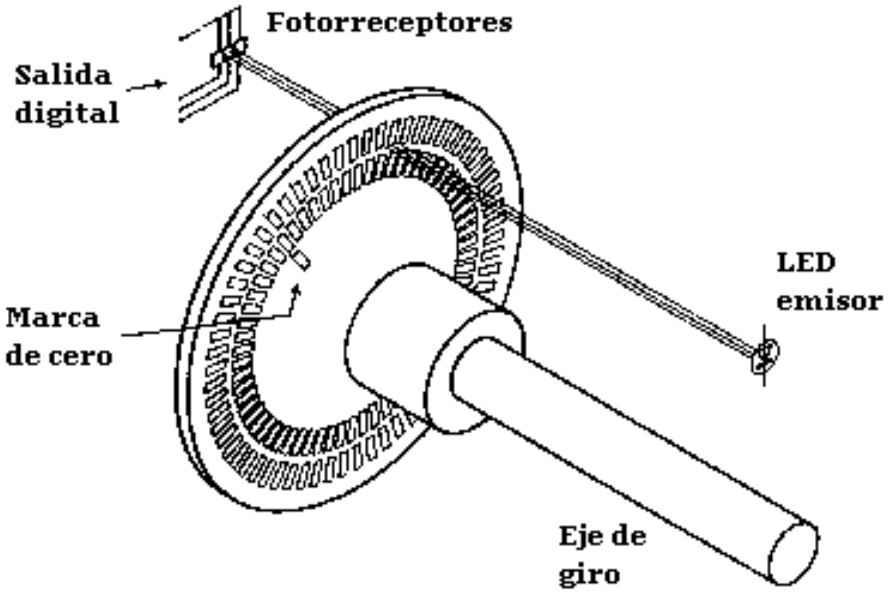


*Figura 1. 26*a. Encoder incremental con fotodiodos y marca de cero [1.11]

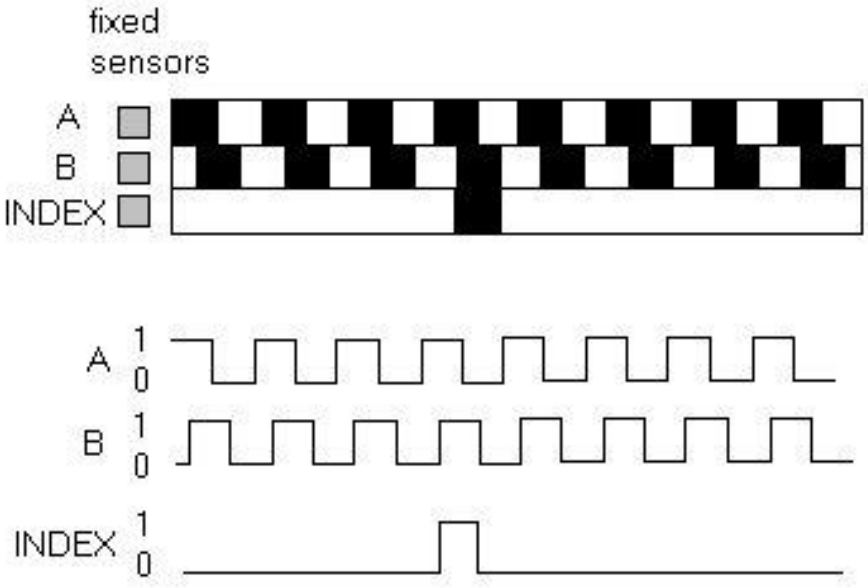


*Figura 1. 26*b. Salida del encoder incremental *[1.12]*

La decodificación de las transiciones A y B se realiza usando circuitos lógicos secuenciales. Estos pueden ofrecer tres diferentes resoluciones 1X, 2X y 4X. La resolución 1X proporciona un pulso de salida en cada flanco negativo (o positivo si se requiere) de la señal A o B, lo que resulta en un solo pulso para cada ciclo o periodo de la señal de cuadratura. La resolución 2X ofrece un pulso de salida en cada flanco positivo y negativo de la señal A o B, lo cual resulta en el doble de pulsos de salida. La resolución 4X ofrece un pulso de salida en cada flanco positivo y negativo de la señales A y B, dando lugar a 4 veces el número de pulsos de salida.

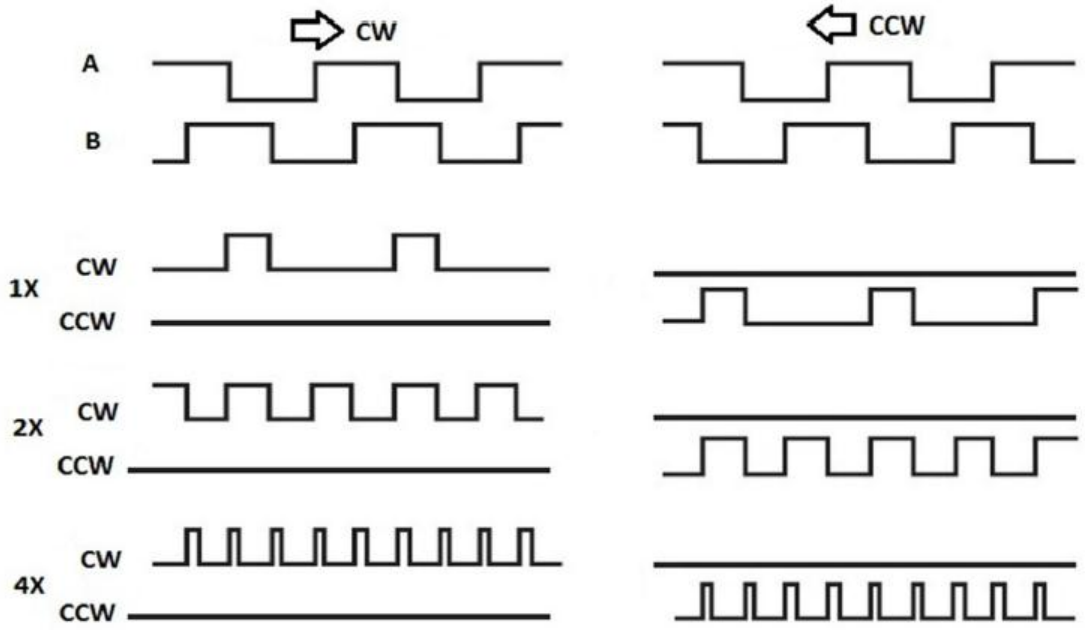


Figura 1. 27. Salidas del encoder incremental y decodificación en modos 1X, 2X, 4X. [1.13]

1.10.1 Determinación del sentido de movimiento.

La información correspondiente al desplazamiento se obtiene directamente de las señales A o B, un ciclo de la señal corresponde al mínimo avance, se puede usar como referencia el flanco de subida o bajada; para un *encoder* de 1600 pulsos por revolución el mínimo avance corresponde a 0.225°. Para determinar la dirección del desplazamiento se requiere de ambas señales (A y B); en la figura 1.28 se tiene un circuito a través del cual se determina el sentido del desplazamiento.

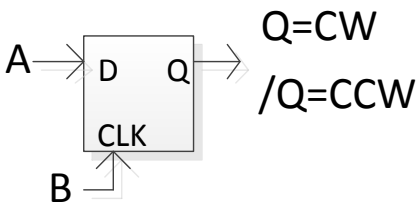


Figura 1. 28. Decodificador de dirección modo 1X [1.14]

La decodificación más básica Se implementa con un *Flip-Flop* "D", la señal A se emplea como dato y B como señal de reloj, en el sentido CW (izquierda a derecha) se captura continuamente un nivel alto, esto, porque el flanco de subida de B coincide con el nivel alto de A ver figura de las señales de cuadratura en la figura 1.29b. Para el sentido CCW (derecha a izquierda) el flanco de subida de B coincide ahora con el nivel bajo de A.

1.10.2 Decodificación en modo 4X.

Para realizar la decodificación en modo 4x se debe tener un reloj que sea mucho más rápido que las señales que se recibirán atreves del encoder con el fin de obtener un muestreo eficiente.

Con estas consideraciones el circuito tradicional para la decodificación se muestra en la figura 1.29

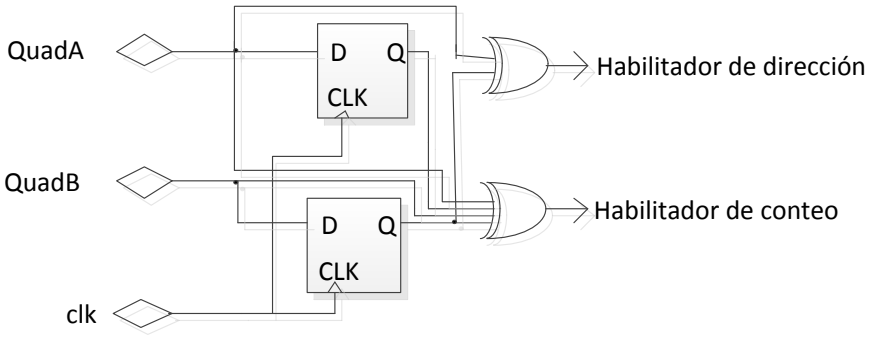


*Figura 1. 29.*Circuito para decodificación en modo 4X. [1.14]

El clk de los FF's es el que proviene de la tarjeta FPGA 50MHz. La simulación de dicho circuito tiene como resultado que la variable "count_enable(CE)" tenga una salida de 1 lógico 4 veces en un ciclo de la señal "quadA" proporcionando la resolución 4x.

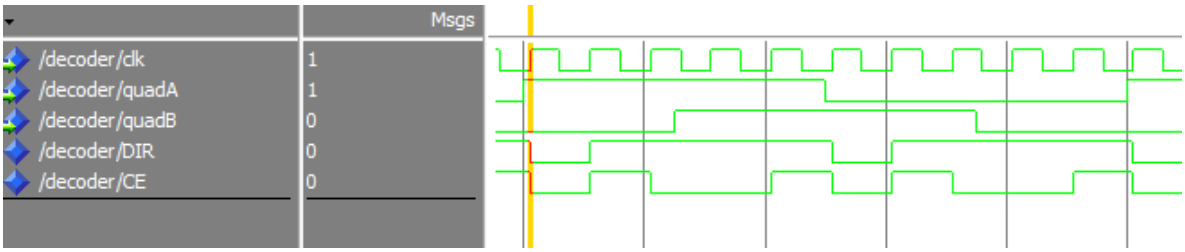


Figura 1. 30. Comportamiento del decodificador 4X

Mientras que el identificador de dirección "count_direction(DIR)" coincide en nivel alto del "count_enable(CE)" mientras el canal A adelanta al canal B , CW figura 1.30.
Este circuito es fácilmente programable en VHDL sin embargo en la práctica las señales en cuadratura no son síncronas con el reloj de la FPGA, la solución clásica es la integración de 2 flip flop D extras con el fin de agregar estabilidad al circuito, con lo que se obtiene el circuito de la figura 1.31.

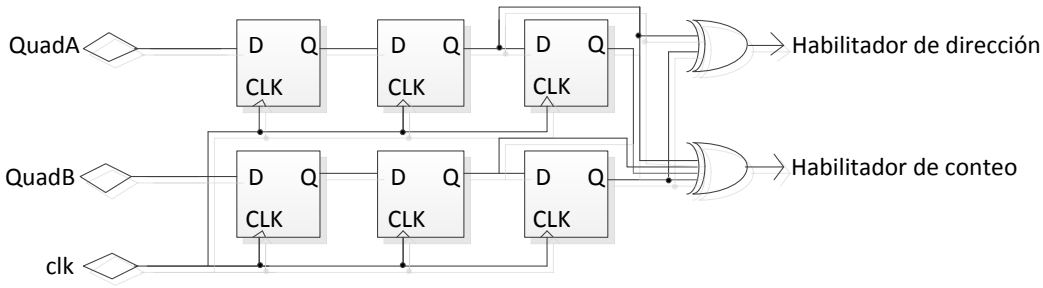


Figura 1. 31. Decodificador de cuadratura. [1.14]

1.10.3 Obtención de la posición del eje a partir de un encoder incremental.

Mediante el habilitador del contador CE, se realiza un conteo de pulsos los cuales están directamente relacionados con el avance en grados del eje del motor, por ejemplo si originalmente se tenían 1600 PPR por canal con el modo 4x Ahora se tienen 6400 PPR, el avance en grados para esta resolución es:

$$Avance = \frac{\#\ 1\ REV\ (Grados)}{PPR} = \frac{360^{\circ}}{6400PPR} = 0.05625^{\circ}/pulso \qquad \textit{Ecuación 1.27}$$

Para obtener las referencias de posición se tiene:

PPR del eje secundario del motor: 16 por canal (ver sección 2.5).
Relación de motoreductor: 100:1 (ver sección 2.4)
Grados totales: 360°.

Ya que se tienen 16 pulsos por canal con cada revolución del eje secundario se tendrán entonces

$$PPR\ main = 16PPR\ shaft * 100 = 1600PPR\ main \qquad \textit{Ecuación 1.28}$$

Ya que se utiliza la decodificación en modo 4X por cada pulso del encoder se tendrán 4 pulsos por software, entonces.

$$PPR\ main\ 4X = 1600PPR * 4 = 6400PPR\ main\ 4x \qquad \textit{Ecuación 1.29}$$

Una revolución tiene 360 grados, por lo tanto el número de conteos por grado es:

$$Conteos\ p\ grados\ (CPG) = \frac{6400PPR\ main\ 4x}{360\ grados} = 17.77 \approx 18\ conteos \qquad \textit{Ecuación 1.30}$$

Así, para obtener las referencias deseadas:

$Ref45^{o} = 17.77 * 45^{\circ} = 800\ \ conteos$
$Ref90^{o} = 17.77 * 90^{\circ} = 1600\ conteos$
$Ref180^{o} = 17.77 * 180^{\circ} = 3200\ conteos$
$Ref220^{o} = 17.77 * 220^{\circ} = 3909\ conteos$

1.10.4 Obtención de la velocidad a partir de un encoder incremental.

Para la estimación de la velocidad se utilizará el método T, dicho método acoge su nombre ya que utiliza un periodo (T) de un pulso proporcionado directamente por el encoder ver figura 1.32. El método requiere de una señal con una frecuencia mucho más alta que la de las señales de cuadratura.

Una estimación de la frecuencia del tren de pulsos del encoder a cierta velocidad se puede desarrollar como sigue:

Si la velocidad máxima del rotor en el cual está el encoder es de 10000rmp, entonces se obtienen las revoluciones por segundo:

$$rps = \frac{10000rpm}{60} = 166rps\ aprox \qquad \textit{Ecuación 1.31}$$

Con este dato y con la cantidad de PPR del encoder se obtienen los pulsos por segundo:

$$PPS = 166rps * 16PPR = 2666pps \qquad \textit{Ecuación 1.32}$$

Por definición la frecuencia es el número de veces que un evento se repite en un segundo, así bajo esta definición por un canal del encoder se tienen 2666 pulsos en un segundo y por lo tanto la frecuencia es de 2666Hz, la frecuencia del reloj que se tienen en la tarjeta cyclone V es de 50MHz por lo que se da por hecho que se tienen los requisitos para aplicar el método.

La velocidad en RPM puede ser calculada como sigue:

$$Nrpm = \frac{60*fc}{m*PPR} \qquad \textit{Ecuación 1.33}$$

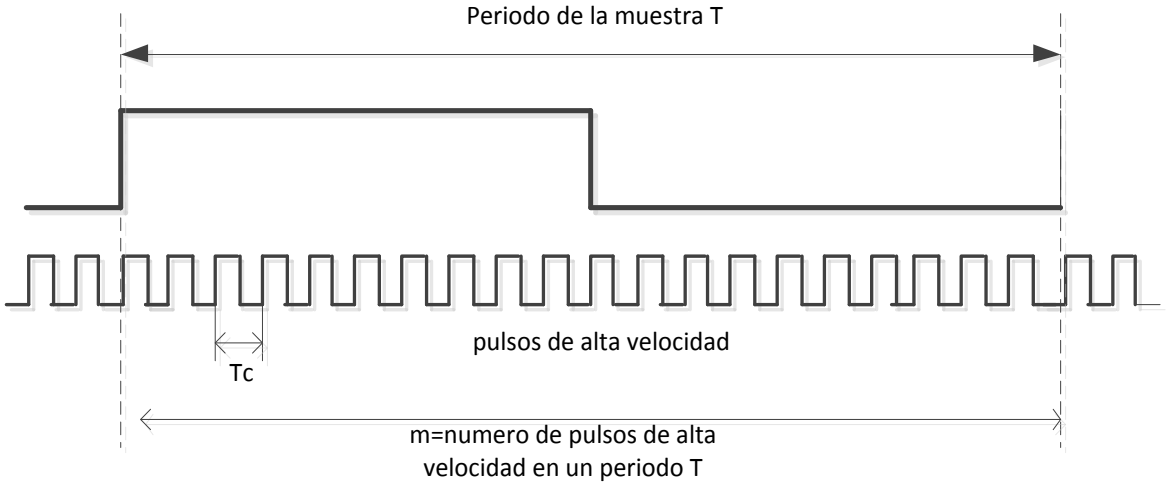


Figura 1. 32 elementos necesarios para el cálculo de velocidad con el método T [1.13]

Dónde:

fc: es la frecuencia del tren de pulsos que se utiliza para medir la velocidad entre pulsos adyacentes provenientes del encoder.

m: es el número de pulsos del tren de alta velocidad que suceden entre pulsos adyacentes.
PPR: los pulsos por revolución del encoder.

Se debe considerar que si el motor tiene un juego de engranes (es un motoreductor) y el encoder está acoplado al eje secundario, entonces habrá que multiplicar por la relación respectiva para obtener la velocidad en el eje principal, así mismo es preferible utilizar el método con resolución 1x.

Este método T puede ser invertido con la finalidad de que el decodificador no se vea afectado por glitch provenientes del encoder. En la figura 1.32 si se establece que T es el tiempo mientras se estará contando los pulsos provenientes del encoder (ahora el tren de alta velocidad son los pulsos de un canal de cuadratura) se tiene el cálculo de las rpm como sigue:

Velocidad Máxima: 100rpm Main.
Relación de motoreductor: 100:1 (ver sección 2.4).
Velocidad Máxima Shaft: 10000rpm (ver sección 2.4).

Cálculo de la frecuencia del tren de pulsos del encoder:

Para 50rpm del main:

$$50rpm\ main = 5000\ rpm\ Shaft$$
$$rps = \frac{5000rpm}{60s} = 83.333rps$$
$$Pulsos\ por\ segundo(pps) = \ 83.33rps * 16pulsos = 1333pps = 1333Hz$$

Si se escoge un periodo T=0.1 segundos se tendrán: 1333pps*0.1 = 133 pulsos del encoder que es la referencia. Similarmente para otras velocidades se tiene:

50rpm=133 pulsos.
60rpm=160 pulsos.
70rpm=187 pulsos.
80rpm=213 pulsos.
90rpm=240 pulsos.

1.11 Comunicación serial.

La comunicación serial es un protocolo muy común entre dispositivos electrónicos y las computadoras, es muy utilizada en aplicaciones de instrumentación para la transmisión y recepción de datos lo que conforma un sistema de adquisición. La comunicación serial se utiliza para transmitir datos en formato American Standard Code for Information Interchange (ASCII por sus siglas en inglés), para realizar esta comunicación se utilizan 3 líneas de transmisión de datos: tierra, transmitir, recibir. Las características más importantes de la comunicación serial son: *velocidad de transmisión de datos, bits de inicio y de parada, bits de datos,* para que exista una comunicación serial los dos puertos a conectarse deben transmitir/recibir a la misma velocidad y en general tener características iguales.

La comunicación que se utiliza para enviar datos desde la tarjeta FPGA cyclone V GX Starter hacia la computadora se realiza es a través del puerto Transmisor-Receptor Asíncrono Universal (UART por sus siglas en inglés) que posee dicha tarjeta. El controlador UART es el componente clave del sistema de comunicación entre una PC y cualquier dispositivo electrónico que posea dicho puerto. Su

función principal es convertir los datos serie a paralelos cuando se trata de datos recibidos (de entrada) y de convertir datos paralelos a serie para transmisión (de salida), toma bytes de datos y transmite los bits individuales en una forma secuencial, en la recepción de estos datos a través de otro puerto o controlador UART reensambla los bits en bytes completos, en la figura 1.33 se observa un esquema general de los bloques básicos de un UART, en esta figura se observan los registros de datos (transmisión y recepción), sus registros de desplazamiento (RxD y TxD), sus registros de control de transmisión y recepción y las señales de sincronización, ya sea para el comienzo de la transmisión o recepción (RTS, CTS).

El UART sirve como base para muchos protocolos de comunicación como lo es el RS-232 (utilizado en los puertos COM), la base de éste es un canal de comunicación de una sola vía en el que en un extremo se transmite y en el otro extremo se recibe. Cuando la línea está libre, está a un nivel lógico alto. La velocidad a la cual se envían los bits se le llama *velocidad de transmisión y esta especificada en BAUDIOS (bit/segundo).* Las velocidades de transmisión más comunes son: 9600, 19200, 38400, 115200 [BAUDIOS] entre otras.

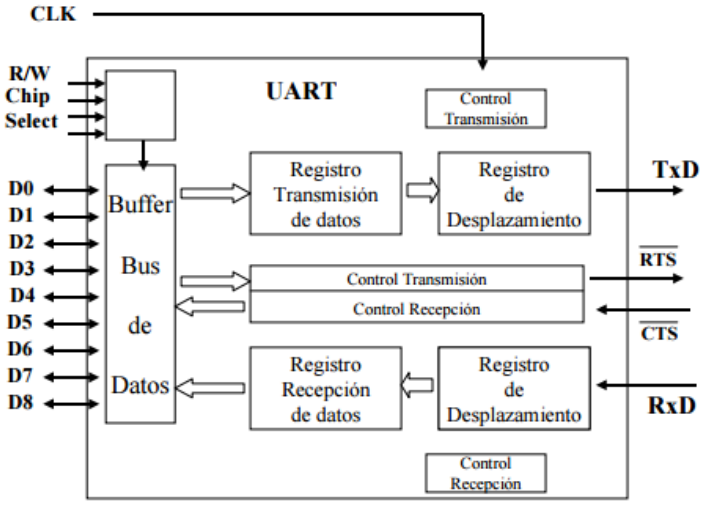


Figura 1. 33 Diagrama de bloques del puerto transmisor-receptor asíncrono universal UART [1.16]

Los parámetros que se han de definir para la correcta utilización del controlador UART son los siguientes:

- La sincronización entre el receptor y el transmisor.
- Codificación de los datos.
- Prioridad en el envío de los bits.
- Tasa o velocidad de envío de datos.

Para la sincronización de la transmisión de los datos se pone un bit de inicio (un bit lógico 0), se envían los 8 bits (uno a uno) empezando por el bit menos significativo, luego se envía un bit de parada (un bit lógico 1) esto se realiza cada vez que se completa una transmisión de datos.

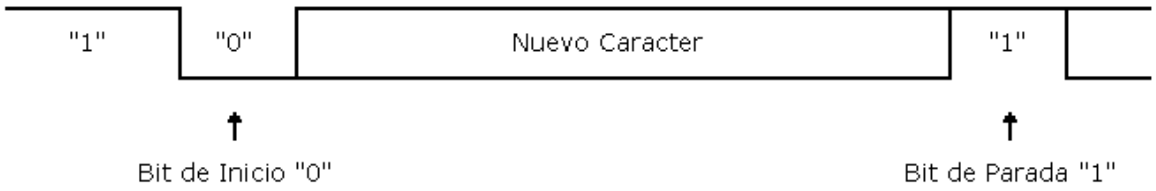


*Figura 1. 34*Formato de transmisión asíncrona. [1.17]

La codificación puede ser cualquier código binario. El más utilizado por los programadores normalmente es el código ASCII que utiliza 7 bits para codificar 96 caracteres imprimibles y 32 caracteres de control.

La tasa de transmisión o también conocida como velocidad de transmisión se mide en baudios o números de bits que se transmiten por segundo (bps), como se mencionó anteriormente, la velocidad pueden ser: 110, 150, 300, 600, 900, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 31250, 38400 y 57800.

El puerto UART usa algunos protocolos o estándares de comunicación de acuerdo a las necesidades que el usuario requiera, entre los que se puede mencionar se encuentran: RS-232, RS-422 y RS-485, en esta ocasión se utiliza el protocolo RS-232 porque ese es el chip que posee la tarjeta FPGA para la comunicación de datos seriales vía UART.

Este protocolo (RS232) es una norma mundial que rige la comunicación serial, por este protocolo se estandarizaron las velocidades de transferencia de datos, la forma en que se transmiten los datos, entre otras características. Al principio, esta norma fue definida para conectar una computadora y un modem, luego se empezó a utilizar para la comunicación serie entre dispositivos en general y la computadora.

Señales para el uso del protocolo RS232.

Petición para enviar o Request to send (RTS). Esta señal se envía a la computadora para indicar que se requiere transmitir datos, una vez que esta señal se activa la computadora o dispositivo electrónico estará listo para enviar datos.

Terminal de datos preparada o Data Terminal Ready (DTR). Esta línea de señal es afirmada por la computadora e informa que está lista para recibir datos.

Transmisión de datos o Transmit Data (TxD). Es acá donde se transmite un bit a la vez.

Recepción de datos o Receive Data (RxD). Es la línea donde se recibe un bit a la vez.

Capítulo II: Hardware para implementación del controlador PID.

Introducción al capítulo II.

En este capítulo se introduce al hardware particular con el que se llevara a cabo la implementación del controlador PID. El dispositivo programable FPGA será la tarjeta cyclone V GX starter de Altera la cual cuenta con los periféricos para la introducción de los datos necesarios para el funcionamiento del controlador así como también los necesarios para la presentación de indicadores y comunicación con la computadora.

El motor particular será un motor de corriente continua el cual cuenta con una caja reductora de modo que posee una velocidad lenta pero buen torque a su salida. Este motor posee un encoder incremental de cuadratura acoplado a su eje secundario el cual provee de 16 pulsos por canal para una revolución del rotor secundario, la importancia del encoder y su resolución radica en la exactitud del giro del motor. Para obtener más exactitud en cuanto al movimiento del motor se realiza un multiplicador de los pulsos por revolución del eje.

2.1 Tarjetas FPGA.

Cuando se trabaja con dispositivos electrónicos, es necesario conocer cómo trabajan y actúan dependiendo de las señales que el usuario le provea al dispositivo. Una parte fundamental de todo proyecto es el conocimiento y el comportamiento del hardware con el cual se trabaja.

Entre otras cosas, se define el hardware como:

Conjunto de elementos físicos y materiales que componen cualquier dispositivo electrónico tales como ordenadores, smartphone, televisores, reproductores de música, robots o placas de desarrollo entre otros ejemplos.

El hardware contiene elementos esenciales de los cuales depende el funcionamiento fundamental del dispositivo electrónico en el cual se trabaja, para una computadora, por ejemplo son: la *placa base* (también conocida como *Placa Madre*), que es una placa de circuito impreso que aloja a la Unidad Central de Procesamiento (CPU) o microprocesador, Chipset (circuito integrado auxiliar), Memoria RAM, BIOS o Flash-ROM, etc., además de comunicarlos entre sí.

A continuación se define *brevemente* las partes fundamentales que componen una tarjeta madre:

Dispositivos de entrada: son aquellos a través de los cuales se envían datos externos a la unidad central de procesamiento, como el teclado, ratón, escáner, entre otros.

Chipset: es uno de los dispositivos más importantes para el funcionamiento del hardware en cuestión, está integrado en la placa base y su función principal es permitir la comunicación entre la unidad central de procesamiento (CPU) y el resto de componentes de la placa base, a través de medios los cuales se les denomina buses, estos son: Northbridge (puente norte) y Southbridge (puente sur).

El puente norte al igual que el puente sur es un circuito integrado el cual hace el puente de enlace entre la comunicación del microprocesador y la/las memorias, así también entre la comunicación con el puente sur.
El puente sur es el que coordina los dispositivos de entrada y salida además de algunas otras funciones de baja velocidad. Este puente se comunica con el CPU a través del puente norte.

Unidad central de procesamiento (CPU): es un conjunto de circuitos integrados que puede estar compuesta por uno o varios microprocesadores que se encargan de interpretar y ejecutar las instrucciones, administrar, ejecutar y procesar datos, con el mundo exterior, es el cerebro de todo hardware.

Unidad de control: es la que se encarga que las instrucciones se ejecuten en memoria, interpretándolas y que luego, serán ejecutadas en la unidad de proceso.
Unidad lógica aritmética (ALU): es la unidad en al cual se llevan a cabo las instrucciones aritméticas y lógicas.
Unidad de almacenamiento: es la que guarda todos los datos en memoria y se divide en memoria principal y memoria secundaria o auxiliar.
Memoria principal (RAM): son circuitos integrados que almacenan los programas, datos y resultados que se están ejecutando por la CPU de forma temporal, pues el contenido de estos se pierde cuando se interrumpe el flujo de energía. La memoria de acceso aleatorio (RAM), se llama así, porque puede acceder a cualquier posición de memoria sin necesidad de seguir un orden específico. Esta puede ser leída y escrita por lo que su contenido puede ser modificado.
La memoria de solo lectura (ROM) viene en chips al igual que la memoria RAM la diferencia de esta, es que contiene una serie de programas por el fabricante de hardware y es sólo de lectura, por lo que no puede ser modificada y tampoco se altera por cortes de flujo de energía. En esta memoria se almacenan los valores correspondientes a las rutinas de arranque o inicio del sistema y a su configuración.
La *memoria caché* o *RAM Caché* es una memoria auxiliar de alta velocidad, que no es más que una copia de acceso rápido de la memoria principal almacenada en los módulos de *RAM*.

Memoria secundaria: son todos los dispositivos en los cuales pueden extraerse datos almacenados en la CPU o las memorias ya que quedan guardados en estos aunque se interrumpa el flujo de energía.

Dispositivos de entrada y salida: son todos aquellos con los cuales el usuario interactúa con el CPU y el mundo exterior, que pueden ser muchos, ejemplos de estos: pantallas LCD, motores de baja corriente, micrófonos, audífonos, luces, etc. Existen también dispositivos bidireccionales que funcionan como dispositivos tanto de entrada como de salida, que pueden ser, memorias USB, CD, DVD, tarjetas de red, etc.

A continuación se explica brevemente el hardware de la tarjeta cyclone V GX starter kit, en la primera sección se mencionan sus componentes y se muestra el funcionamiento de algunos componentes antes de programarlos, esto es posible gracias al panel de control que viene integrado en un CD junto con la tarjeta. La siguiente sección presenta como se puede utilizar cada componente de la tarjeta y cuáles son los pines para usar estos componentes, ya que es importante tomar en cuenta esto a la hora de programar y ejecutar circuitos con la tarjeta.

2.1.2 Conociendo la tarjeta de desarrollo Field Programable Gate Array (FPGA) Cyclone V GX Starter Kit.

La tarjeta Cyclone V GX Starter Kit presenta una plataforma de diseño de hardware robusto construido por Altera Cyclone V GX. La tarjeta de desarrollo Cyclone V GX Starter Kit incluye hardware para puertos de arduino, USB-Blaster, GPIO, USB-UART, audio, video y mucho más.

Esta placa contiene todos los componentes necesarios para usarla con una computadora que contenga una versión de Windows XP o superior.

Contenido del paquete
La figura 2.1 muestra el contenido de la tarjeta Cyclone V GX Starter Kit.

Figura 2.1 Contenido de la tarjeta FPGA Cyclone V GX Starter Kit. [2.1]

El contenido de la tarjeta Field Programable Gate Array (FPGA) Cyclone V GX Starter Kit, incluye:

- La tarjeta Cyclone GX Starter Kit.
- Fuente de poder de 12V DC.
- Cable USB Tipo A hembra a Tipo B macho (USB-Blaster).
- CD de instalación.

CD del Sistema Cyclone V GX Starter Kit.

El CD de instalación de la tarjeta FPGA contiene los materiales de documentación y soporte necesarios, incluyendo, el manual de usuario, panel de control, integrador de sistema, referencias de diseño y hoja de datos de dispositivos.

Diseño y componentes.

Las figura 2.2 y figura 2.3 muestran la ubicación de los puertos y componentes claves.

La tarjeta contiene características que permiten al usuario implementar diversos diseños de circuitos, desde circuitos simples hasta proyectos multimedia.

La tarjeta posee el siguiente hardware:

Dispositivo FPGA.

- Dispositivo Cyclone V GX 5CGXFC5C6F27C7N
- 77K de elementos lógicos programables.
- 4884Kbits de memoria de sistemas embebidos.
- Seis PLLs fraccionarios.
- Dos controladores de memoria de disco duro.

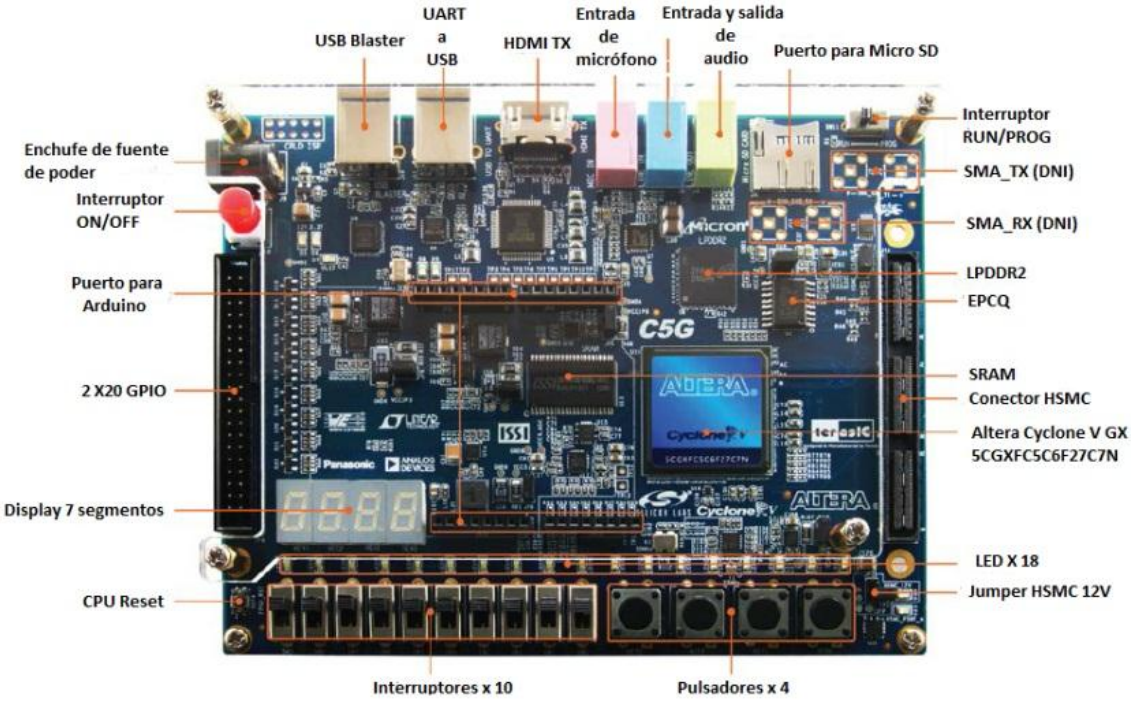


Figura 2.2 Vista superior Tarjeta FPGA Cyclone V GX Starter kit. [2.1]

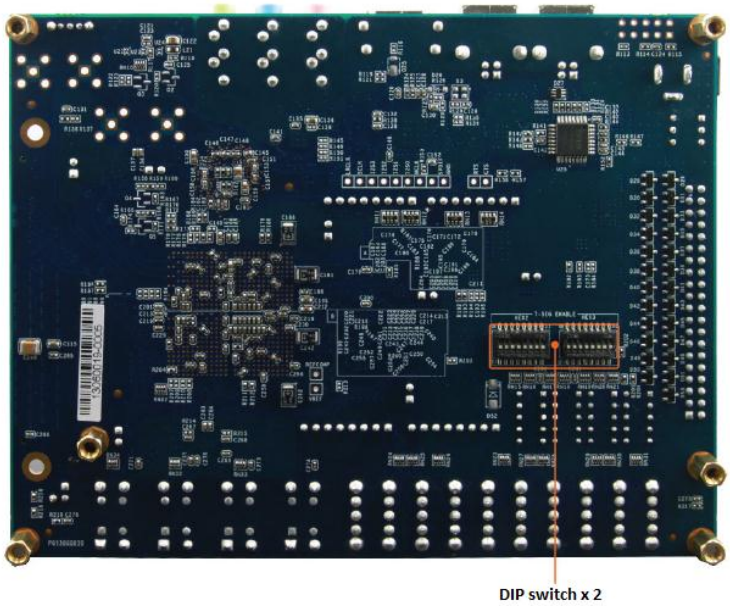


Figura 2.3 Vista inferior Tarjeta FPGA Cyclone V GX Starter Kit. [2.1]

Configuración y depuración.

- Dispositivo de configuración de serie Quad EPCQ256 en FPGA
- USB blaster (conector tipo B)

Dispositivos de memoria:

- Bus de datos LPDDR2 x 32 bits.
- Bus de datos SRAM x 16 bits.

Comunicación.

- UART-USB

Puertos:

- Puertos GPIO 2x20.
- Puerto para arduino, incluyendo pines analógicos.
- HSMC x 1.

Display:

- HDMI TX, compatible con DVI v1.0 y HDCP v1.4

Audio:

- CODEC de 24 bits, salida de audio, entrada de audio y salida de micrófono.

Interruptores, pulsadores y leds:

- 18 LEDs.
- 10 interruptores deslizantes.
- 4 pulsadores con anti rebote.
- 1 pulsador de reinicio de CPU.

2.1.2 Diagrama de bloques de la tarjeta Cyclone V GX Starter Kit.

La figura 2.4 muestra el diagrama de bloques de conexión de la tarjeta. Todas las conexiones se realizan a través del dispositivo Cyclone V GX, por lo tanto el usuario puede configurar la FPGA para implementar el diseño de cualquier sistema.

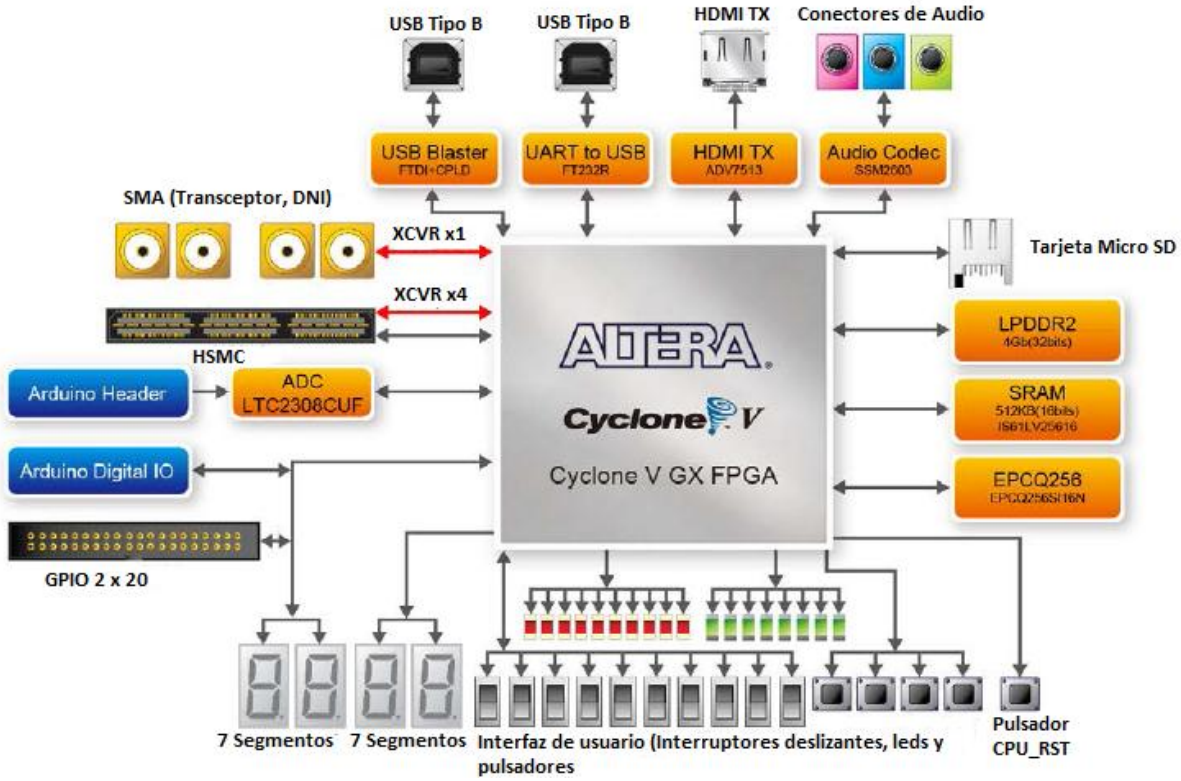


Figura 2.4 Diagrama de Bloques de conexión entre puertos E/S y chip Cyclone V GX FPGA. [2.1]

2. 2 Uso de los componentes de la tarjeta.

Configuración, estado e instalación.

La tarjeta cyclone V GX Starter Kit contiene un dispositivo de configuración serie que almacena los datos de configuración para el chip Cyclone V GX. Estos datos de configuración se cargan automáticamente desde el dispositivo de configuración en la FPGA cuando se enciende. Usando el software Quartus II, es posible configurar la FPGA en cualquier momento, y también es posible cambiar los datos no volátiles que se almacena en el dispositivo de configuración de serie. A continuación se describen dos tipos de métodos de programación.

1. ***Programación Joint Test Action Group (JTAG):*** En este método de programación, llamado así por el JTAG estándares IEEE, el flujo de bits de configuración se descarga directamente en el chip cyclone V GX. La FPGA conservará esta configuración, siempre y cuando se aplica energía a la placa; la información de configuración se perderá cuando la energía se interrumpa.

2. ***Programación Active Serial (AS):*** En este método, llamado de programación de serie activa, el flujo de bits de configuración se descarga en el dispositivo de configuración serie *Altera EPCQ256*. Proporciona almacenamiento no volátil del flujo de bits, por lo que la información se mantiene incluso cuando la fuente de alimentación a la placa del chip cyclone V GX

Starter Kit está apagado. Cuando la alimentación de la tarjeta está activada, los datos de configuración en el dispositivo EPCQ256 se carga automáticamente en el Cyclone V GX.

2.2.1 Modo de programación JTAG en cyclone V GX Starter Kit

Para configurar el dispositivo FPGA en el modo de programación JTAG, el interruptor JTAG en la tarjeta cyclone V GX Starter Kit debe estar en la posición *RUN* (se puede leer la posición en la tarjeta) para que permita al programa Quartus II detectar los dispositivos FPGA. La figura 2.5 muestra la ubicación del interruptor JTAG en la tarjeta cyclone V GX Starter Kit. En el circuito los pines denominados pin1 y pin2 en JP2 puede desactivar las señales JTAG en el conector HSMC que formarán una cadena de bucle cerrado JTAG en la tarjeta cyclone V GX Starter Kit (Ver Figura 2.6). Por lo tanto, sólo el dispositivo FPGA conectado será detectado por el programa Quartus II. Si los usuarios quieren incluir otro dispositivo FPGA o interfaz que contenga dispositivos FPGA a través del conector HSMC, retire el jumper del pin1 y pin2 de JP2 para permitir a los puertos de señal JTAG comunicarse con el conector HSMC.

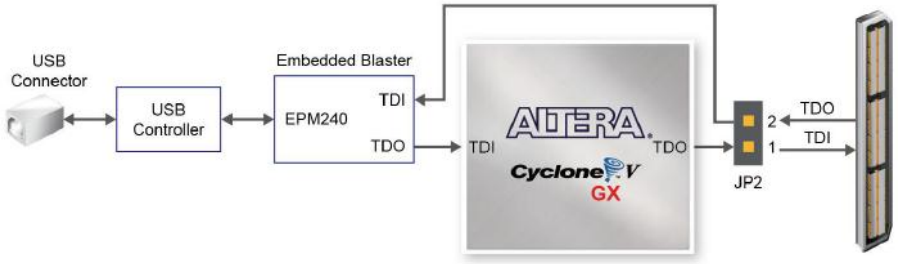


Figura 2.5. Diagrama de conexión interna entre el pin denominado JP2 para modo de programación JTAG y el chip cyclone V GX para interactuar con el puerto HSMC. [2.1]

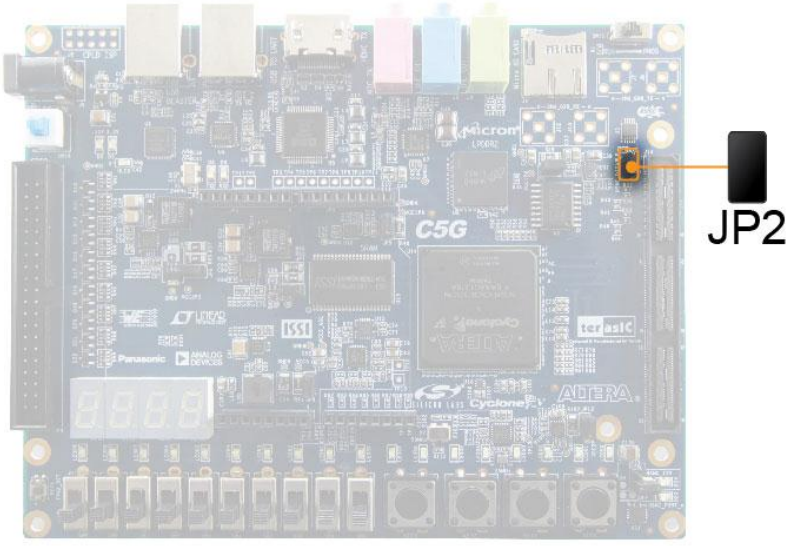


Figura 2.6 Ubicación del pin JP2 en la tarjeta cyclone V GX Starter Kit. [2.1]

A continuación se describen los pasos a seguir para operar la tarjeta en los modos de programación *JTAG* y *AS*. Para ambos modos de programación la tarjeta cyclone V GX Starter debe estar conectado a un ordenador mediante el cable USB-Blaster.

Configuración de la FPGA en el modo de programación JTAG

La figura 2.7 ilustra la configuración del modo *JTAG.* Para descargar una configuración en la tarjeta cyclone, es necesario realizar los siguientes pasos:

- Asegurar la aplicación energía a la tarjeta a través de la fuente de poder.
- Configurar el modo de programación colocando el interruptor *RUN/PROG* (denominado también como SW11) en la posición *RUN* (ver figura 2.8).
- Conectar el cable USB-Blaster a la tarjeta cyclone V GX Starter Kit.
- Presionar el botón *Power* de la tarjeta, y a partir de este momento la tarjeta puede ser usada para programarla a través de Quartus II, se debe descargar a la tarjeta el archivo de programación con la extensión *.sof* como se explica en el capítulo 3.

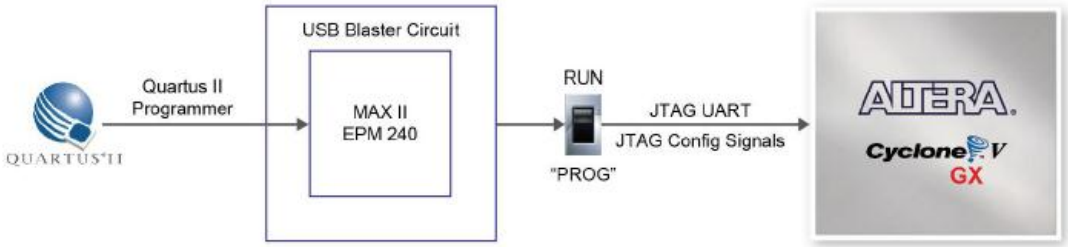


Figura 2.7. Esquema de configuración JTAG. [2.1]

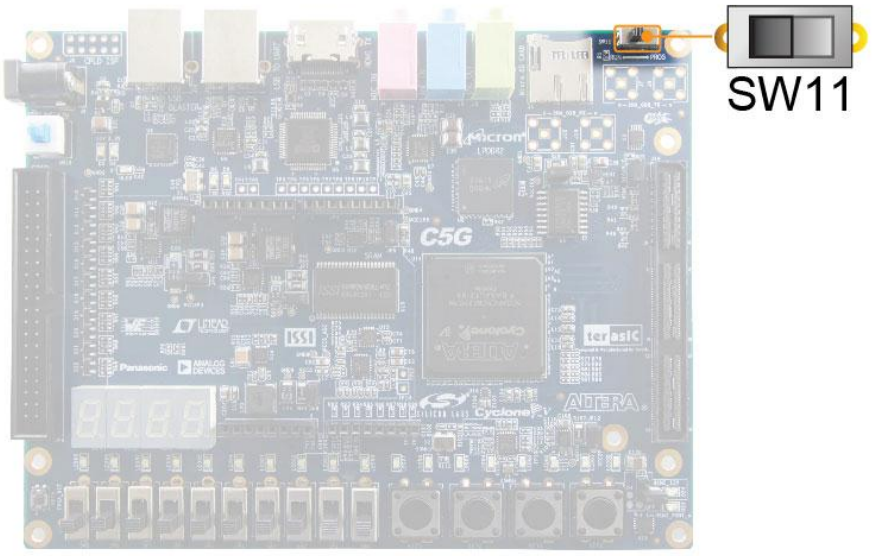


Figura 2.8. SW11 que indica el modo de programación JTAG en la posición RUN, la posición del interruptor se puede observar en la tarjeta [2.1]

Configuración del EPCQ256 en el modo de programación AS.

La figura 2.9 ilustra el esquema de comunicación para el modo de programación *AS*. Para poder usar este modo de programación es necesario seguir los pasos que a continuación se detallan:

- Asegúrese de aplicarle energía a la tarjeta a través de la fuente de poder.
- Conecte el cable USB-Blaster a la tarjeta cyclone V GX Starter Kit.
- Configurar el modo de programación colocando el interruptor *RUN/PROG* (denominado también como SW11) en la posición *PROG*.
- Ahora puede programar el chip EPCQ256 en el modo de programación *AS* a través de Quartus II, en este caso, se debe descargar a la tarjeta el archivo de programación con extensión *.pof*
- Una vez el usuario finalice la operación de programación, ajuste el interruptor *RUN/PROG* de nuevo a la posición *RUN* y luego reiniciar la tarjeta cyclone V GX oprimiendo el interruptor de alimentación y vuelva a encenderla; esta acción hace que los nuevos datos de configuración en el dispositivo EPCQ256 puedan ser cargados en el chip FPGA.

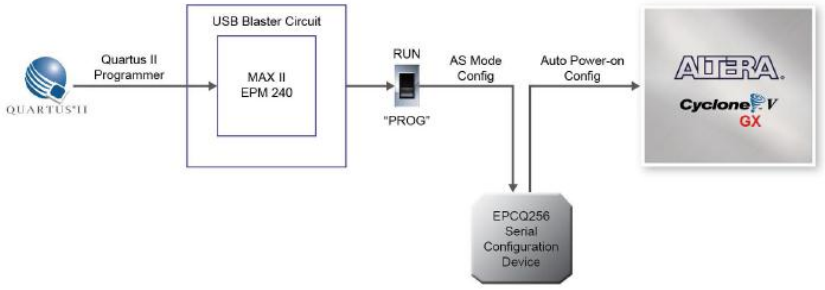


Figura 2.9. Esquema de configuración en el modo de programación AS y como se configura con el software de programación Quartus II [2.1]

Estado de los LED's.

La tarjeta de desarrollo FPGA incluye LED's de estado específicos para indicar el estado de la tarjeta. Consulte la Tabla 2.1 para la descripción de cada indicador LED. Refiérase a la Figura 2.10 para la ubicación detallada en la tarjeta de cada LED.

Referencia en la tarjeta	Nombre del LED	Descripción
D5	Power a 12V	Se ilumina cuando se activan 12V
D6	Power a 3.3V	Se ilumina cuando se activan 3.3V
D24	HSMC_12V-Power	Se ilumina cuando HSMC 12V está activado.
D23	HSMC_PSNT_n	Se ilumina cuando en HSMC está presente una tarjeta secundaria
D7	ULED	Se ilumina cuando en la tarjeta hay un flujo de datos a través del USB-Blaster.

Tabla 2.1 Descripción de los estados de los LED's. [2.1]

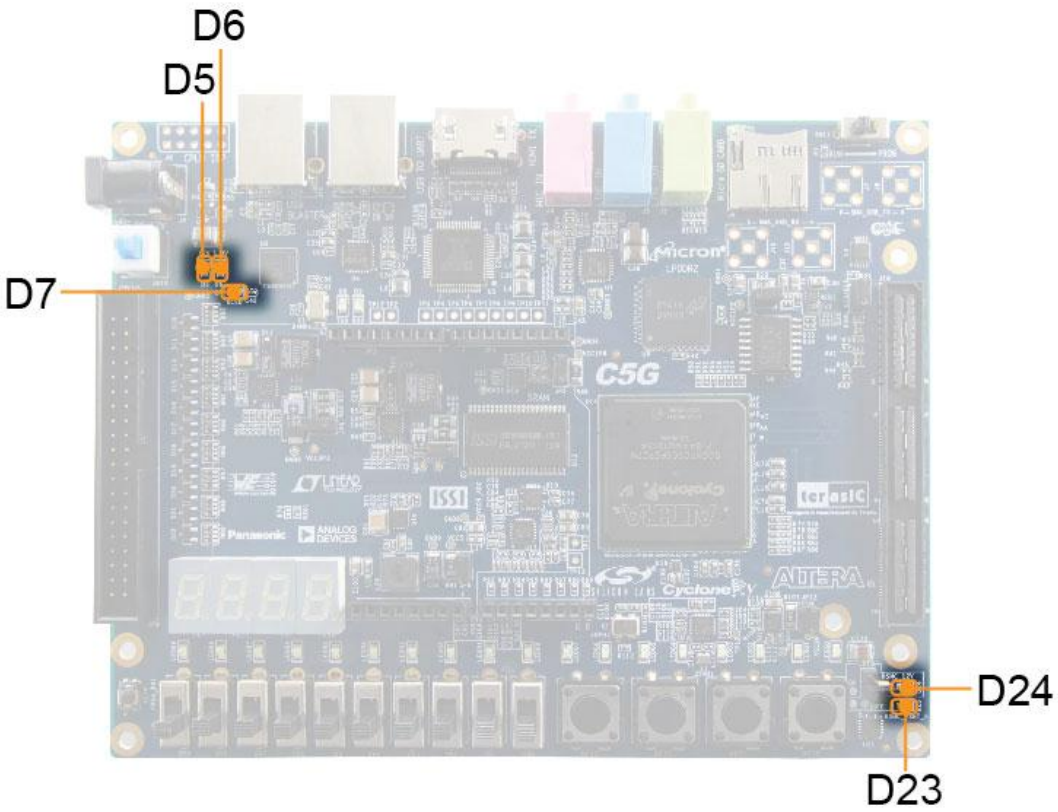


Figura 2.10. Ubicación de los LED's de estado en la tarjeta cyclone V GX starter Kit [2.1]

2.3 Uso general de Entradas/Salida.

2.3.1 Uso de pulsadores definidos para el usuario.

La tarjeta cyclone V GX Starter Kit incluye cuatro pulsadores definidos que permiten a los usuarios interactuar con la tarjeta, como se muestra en la figura 2.11. Cada uno de estos pulsadores contiene un sistema anti-rebote utilizando Schmitt Trigger.

Los pulsadores denominados *KEY0, KEY1, KEY2 y KEY3* se conectan directamente al chip cyclone V GX de la tarjeta FPGA. Cada pulsador proporciona un nivel lógico bajo cuando se presionan y tiene un nivel lógico alto cuando no se presiona, es decir, son pulsadores normalmente abiertos. Como los pulsadores tienen sistema anti-rebote, son ideales para realizar circuitos como relojes o para restablecer las entradas de un circuito, se pueden utilizar según las necesidades que el usuario o programador le convengan.

En la tabla 2.2 se pueden observar las nomenclaturas de los pulsadores y su respectiva descripción para que el usuario tenga conocimiento al momento de programar un circuito.

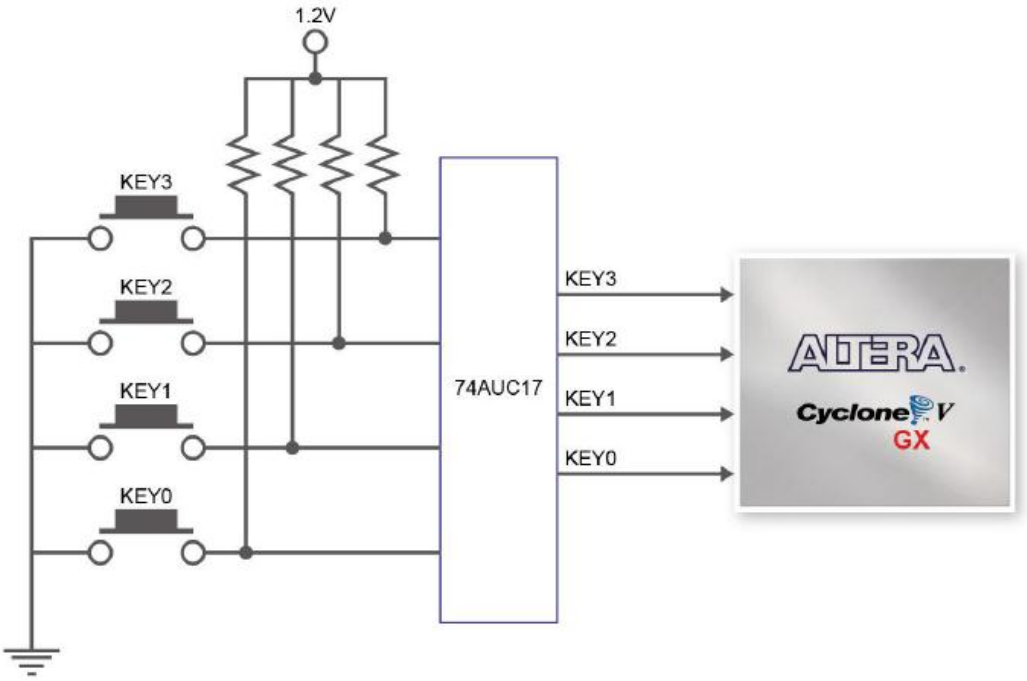


Figura 2.11. Diagrama de conexión interna entre los pulsadores y el chip cyclone V GX. [2.1]

Referencia en la tarjeta	Nombre asignado	Descripción	E/S estándar	Número de pin en el chip cyclone V GX
KEY0	KEY0	Nivel lógico alto cuando el pulsador no está presionado. Los cuatro pulsadores pasan a través del circuito anti-rebotes.	1.2V	PIN_P11
KEY1	KEY1		1.2V	PIN_P12
KEY2	KEY2		1.2V	PIN_Y15
KEY3	KEY3		1.2V	PIN_Y16
KEY4	CPU_RESET_n	Nivel lógico alto cuando el pulsador no está presionado	3.3V	PIN_AB24

Tabla 2.2. Asignación de los pines de los pulsadores en la tarjeta cyclone V GX Starter Kit. [2.1]

2.3.2 Interruptores.

Se disponen de 10 interruptores deslizantes para que el usuario pueda interactuar con la tarjeta.

A diferencia de los pulsadores, estos interruptores no contienen un circuito anti-rebote, entonces, podrán ser utilizados como entradas de nivel sensible a los circuitos. Cada interruptor está conectado a diferentes pines del chip cyclone de la tarjeta FPGA. Cuando el interruptor se encuentra en la posición baja, proporciona un nivel lógico “0” a la FPGA, y cuando el interruptor se encuentra en la posición alta, proporciona un nivel lógico “1”.

En la tabla 2.3 se pueden observar las nomenclaturas de los interruptores deslizantes y su respectiva descripción para que el usuario tenga conocimiento al momento de programar un circuito.

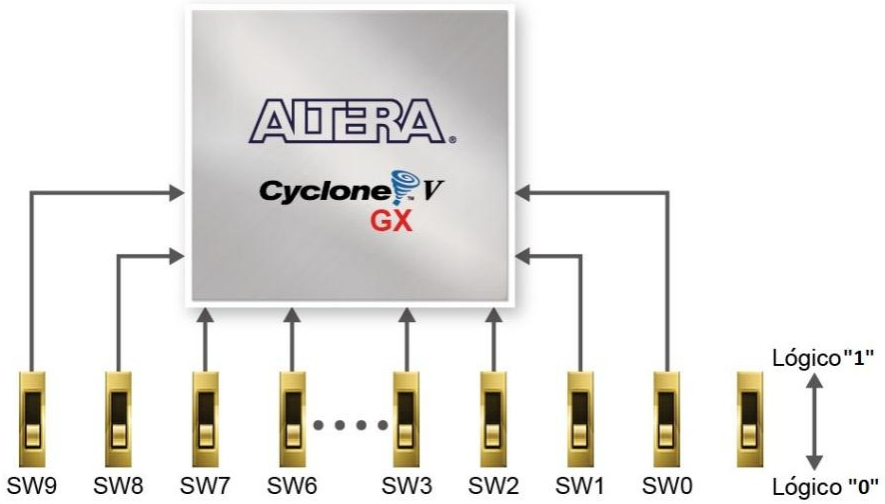


Figura 2.12 Esquema de conexión interna entre los interruptores deslizantes y el chip cyclone V GX. [2.1]

Referencia en la tarjeta	Nombre asignado	Descripción	E/S Estándar	Número de pin en el chip cyclone V GX
SW0	SW0	Interruptor deslizante [0]	1.2V	PIN_AC9
SW1	SW1	Interruptor deslizante [1]	1.2V	PIN_AE10
SW2	SW2	Interruptor deslizante [2]	1.2V	PIN_AD13
SW3	SW3	Interruptor deslizante [3]	1.2V	PIN_AC8
SW4	SW4	Interruptor deslizante [4]	1.2V	PIN_W11
SW5	SW5	Interruptor deslizante [5]	1.2V	PIN_AB10
SW6	SW6	Interruptor deslizante [6]	1.2V	PIN_V10
SW7	SW7	Interruptor deslizante [7]	1.2V	PIN_AC10
SW8	SW8	Interruptor deslizante [8]	1.2V	PIN_Y11
SW9	SW9	Interruptor deslizante [9]	1.2V	PIN_AE19

Tabla 2.3. Asignación de pines de los interruptores deslizantes en la tarjeta cyclone V GX Starter Kit.[2.1]

2.3.3 Leds rojos y verdes.

La tarjeta cyclone también cuenta con 18 led’s que pueden ser controlados por el usuario, estos se dividen en 10 led color rojos y 8 led verdes, los led’s rojos se ubican por encima de los interruptores y los led’s verdes se ubican por encima de los pulsadores. Los leds se encienden con un nivel lógico alto y se apagan con un nivel lógico bajo, es decir, el usuario al programar el uso de cada led

mediante el software de programación puede controlar dichos leds ya sea con un pulsador, interruptor o una señal externa. El esquema de conexión entre los leds y el chip cyclone V GX se observa en la figura 2.13.

Figura 2.13. Conexión interna entre los led's disponibles para el usuario y el chip cyclone V GX. [2.1]

Referencia en la tarjeta	Nombre asignado	Descripción	E/S Estándar	Número de pin en el chip cyclone V GX
LEDR0	LEDR0	Cada led se enciende cuando se recibe un nivel de estado lógico "1" y se apaga cuando recibe un nivel de estado lógico "0"	2.5 V	PIN_F7
LEDR1	LEDR1		2.5 V	PIN_F6
LEDR2	LEDR2		2.5 V	PIN_G6
LEDR3	LEDR3		2.5 V	PIN_G7
LEDR4	LEDR4		2.5 V	PIN_J8
LEDR5	LEDR5		2.5 V	PIN_J7
LEDR6	LEDR6		2.5 V	PIN_K10
LEDR7	LEDR7		2.5 V	PIN_K8
LEDR8	LEDR8		2.5 V	PIN_H7
LEDR9	LEDR9		2.5 V	PIN_J10
LEDG0	LEDG0		2.5 V	PIN_L7
LEDG1	LEDG1		2.5 V	PIN_K6
LEDG2	LEDG2		2.5 V	PIN_D8
LEDG3	LEDG3		2.5 V	PIN_E9
LEDG4	LEDG4		2.5 V	PIN_A5
LEDG5	LEDG5		2.5 V	PIN_B6
LEDG6	LEDG6		2.5 V	PIN_H8
LEDG7	LEDG7		2.5 V	PIN_H9

Tabla 2.4. Asignación de los leds a la tarjeta cyclone V GX Starter Kit. [2.1]

2.3.4 Displays de 7 segmentos.

La tarjeta FPGA contiene 4 displays de 7 segmentos. Como se muestra en la figura 2.14, los displays de 7 segmentos son ánodo común y están conectados a los pines del chip Cyclone V GX. Si se aplica un nivel de estado lógico bajo, a un segmento este se iluminara, si se aplica un nivel de estado lógico alto, este, se mantendrá apagado. Al igual que los leds estos se activan de acuerdo el usuario lo requiera en el proceso de programación, puede hacerse mediante los pulsadores, interruptores e incluso mediante señales externas según el usuario los programe.

Hay que tomar en cuenta que los display de 7 segmentos nombrados como *HEX3* y *HEX2*, comparten un bus de datos con los GPIO. Cuando se usan estos dos displays, es necesario que los interruptores nombrados como *S1/S2* que están localizados en la parte trasera de la tarjeta, se encuentren en la posición ON antes de tomar el control total de estos displays.

Cada segmento en el display es identificado por un número listado del 0 al 6, con las posiciones que se muestran en la figura 2.15. La tabla 2.5 contiene las nomenclaturas de cada display y su respectiva descripción.

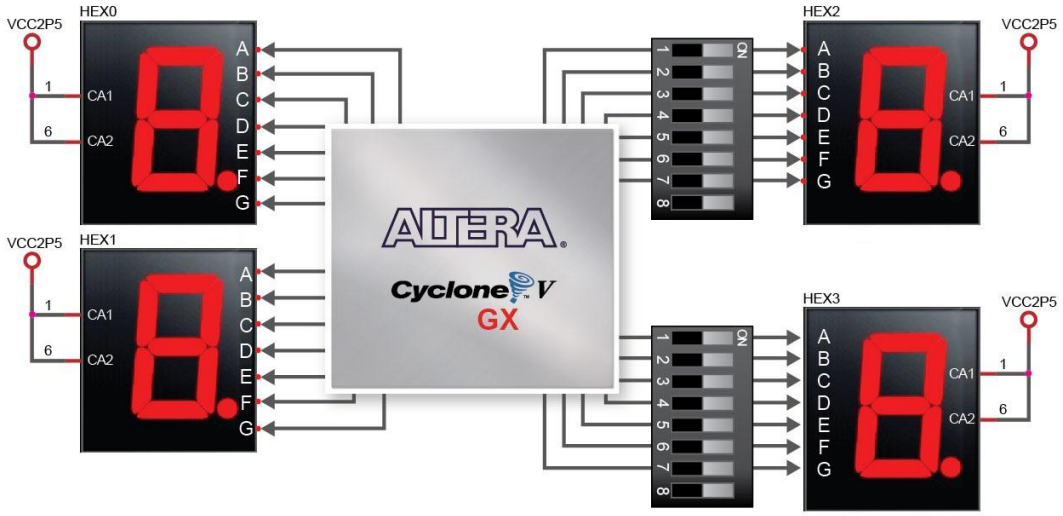


Figura 2.14 Conexión entre los displays y el chip cyclone V GX [2.1]

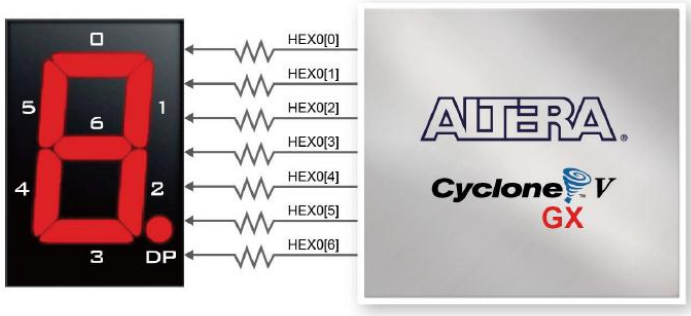


Figura 2.15 Conexión interna entre el display HEX0 y el chip cyclone V GX. [2.1]

Referencia en la tarjeta	Nombre asignado	Descripción	E/S estándar	Número de pin en el chip cyclone V GX
HEX0	HEX0_D0	siete segmentos del digito 0 [0]	2.5V	PIN_V19
HEX0	HEX0_D1	siete segmentos del digito 0 [1]	2.5V	PIN_V18
HEX0	HEX0_D2	siete segmentos del digito 0 [2]	2.5V	PIN_V17
HEX0	HEX0_D3	siete segmentos del digito 0 [3]	2.5V	PIN_W18
HEX0	HEX0_D4	siete segmentos del digito 0 [4]	2.5V	PIN_Y20
HEX0	HEX0_D5	siete segmentos del digito 0 [5]	2.5V	PIN_Y19
HEX0	HEX0_D6	siete segmentos del digito 0 [6]	2.5V	PIN_Y18
HEX1	HEX1_D0	siete segmentos del digito 1 [0]	2.5V	PIN_AA18
HEX1	HEX1_D1	siete segmentos del digito 1 [1]	2.5V	PIN_AD26
HEX1	HEX1_D2	siete segmentos del digito 1 [2]	2.5V	PIN_AB19
HEX1	HEX1_D3	siete segmentos del digito 1 [3]	2.5V	PIN_AE26
HEX1	HEX1_D4	siete segmentos del digito 1 [4]	2.5V	PIN_AE25
HEX1	HEX1_D5	siete segmentos del digito 1 [5]	2.5V	PIN_AC19
HEX1	HEX1_D6	siete segmentos del digito 1 [6]	2.5V	PIN_AF24
HEX2	HEX2_D0	siete segmentos del digito 2 [0], comparte GPIO22	3.3V	PIN_AD7
HEX2	HEX2_D1	siete segmentos del digito 2 [1], comparte GPIO23	3.3V	PIN_AD6
HEX2	HEX2_D2	siete segmentos del digito 2 [2], comparte GPIO24	3.3V	PIN_U20
HEX2	HEX2_D3	siete segmentos del digito 2 [3], comparte GPIO25	3.3V	PIN_V22
HEX2	HEX2_D4	siete segmentos del digito 2 [4], comparte GPIO26	3.3V	PIN_V20
HEX2	HEX2_D5	siete segmentos del digito 2 [5], comparte GPIO27	3.3V	PIN_W21
HEX2	HEX2_D6	siete segmentos del digito 2 [6], comparte GPIO28	3.3V	PIN_W20
HEX3	HEX3_D0	siete segmentos del digito 3 [0], comparte GPIO29	3.3V	PIN_Y24
HEX3	HEX3_D1	siete segmentos del digito 3 [1], comparte GPIO30	3.3V	PIN_Y23
HEX3	HEX3_D2	siete segmentos del digito 3 [2], comparte GPIO31	3.3V	PIN_AA23
HEX3	HEX3_D3	siete segmentos del digito 3 [3], comparte GPIO32	3.3V	PIN_AA22
HEX3	HEX3_D4	siete segmentos del digito 3 [4], comparte GPIO33	3.3V	PIN_AC24
HEX3	HEX3_D5	siete segmentos del digito 3 [5], comparte GPIO34	3.3V	PIN_AC23
HEX3	HEX3_D6	siete segmentos del digito 3 [6], comparte GPIO35	3.3V	PIN_AC22

Tabla 2.5. Asignación de pines de los display de 7 segmentos a la tarjeta cyclone V GX Starter Kit. [2.1]

2.3.5 Circuito de Reloj.

La tarjeta FPGA incluye un generador de reloj de 50 MHz de frecuencia y uno programable.

La programación del generador de reloj es muy flexible. El generador de reloj es controlado por la tarjeta FPGA a través del interface serial I2C. Se puede modificar la frecuencia entre 0.16 y 200 MHz. La tabla 2.6 lista las fuentes de reloj, nombre de señales, frecuencias y los pines correspondientes del chip cyclone V GX. La tabla 2.7 lista el control de pines del generador de reloj, nombre de la

señales, E/S estándar y los pines correspondientes al chip cyclone V GX. Es importante destacar que la frecuencia de ellos viene preestablecida, por ejemplo: al escribir la palabra reservada en el software de programación *CLOCK_50_B3B* se elige uno de los relojes con frecuencia de 50 MHz y así sucesivamente para los demás como lo muestra la tabla 2.6.

Fuente	Nombre de la señal	Frecuencia	E/S estándar	Número de pin en el chip cyclone V GX
X2	CLOCK_50_B3B	50.0 MHz	1.2 V	PIN_T13
U20	CLOCK_125_p	125.0 MHz	0-2.4V	PIN_U12
U20	CLOCK_125_n	125.0 MHz	0-2.4V	PIN_V12
X2	CLOCK_50_B5B	50.0 MHz	3.3 V	PIN_R20
	CLOCK_50_B6A	50.0 MHz	3.3 V	PIN_N20
U20	CLOCK_50_B7A	50.0 MHz	2.5 V	PIN_H12
U20	CLOCK_50_B3A	50.0 MHz	2.5 V	PIN_M10
U20	REFCLK_p0	125.0 MHz	1.5 V	PIN_V6
U20	REFCLK_n0	125.0 MHz	1.5 V	PIN_W6
U20	REFCLK_p1	156.25 MHz	1.5 V	PIN_N7
U20	REFCLK_n1	156.25 MHz	1.5 V	PIN_P6

Tabla 2.6. Asignación de pines para circuito de reloj que contiene la tarjeta cyclone V GX Starter. [2.1]

Oscilador Programable	Nombre de la señal	E/S estándar	Número de pin en chip V GX	Descripción
U 20 (Si5338)	I2C_SCL	2.5 V	PIN_B7	El bus I2C está directamente relacionado con Si5338
	I2C_SDA	2.5 V	PIN_G11	

Tabla 2.7 Asignación de pines del control de reloj programable. [2.1]

2.3.6 Interfaz serial RS-232-USB.

El puerto RS-232 está diseñado para realizar la comunicación entre la tarjeta FPGA y una computadora. Permite una velocidad de transmisión de hasta 3 Mbps. La interfaz se realiza mediante un puerto UART-USB que contiene un chip FT232R y que se conecta a un equipo mediante un conector USB tipo B. La figura 2.16 muestra los esquemas relacionados y la tabla 2.8 se muestra la asignación de pines del RS-232, en la tabla 2.9 se muestra los leds de estado del puerto RS-232.

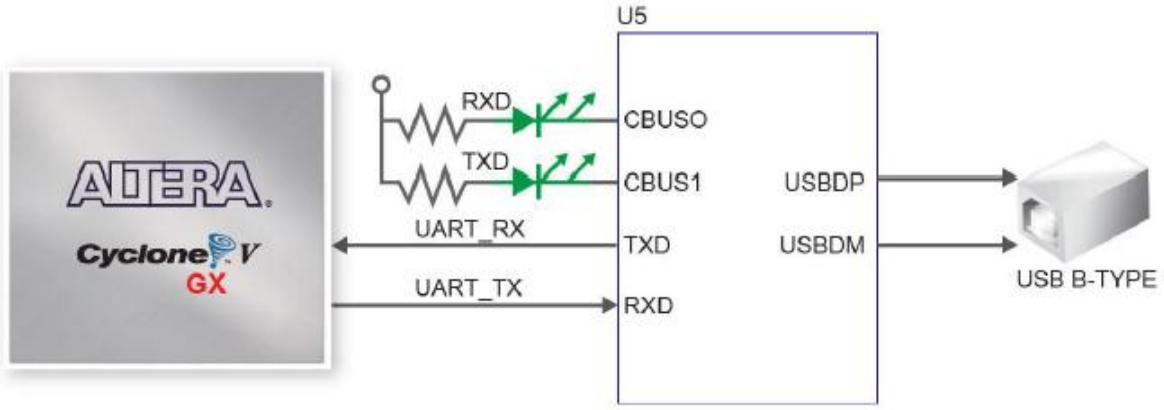


Figura 2.16. Conexión interna entre el chip cyclone V GX y el chip FT232R. [2.1]

Referencia en la tarjeta	Descripción	E/S estándar	Número de pin en cyclone V GX
UART_TX	Salida asíncrona de transmisión de datos	2.5 V	PIN_L9
UART_RX	Entrada asíncrona de receptor de datos	2.5 V	PIN_M9

Tabla 2.8 Asignación de pines para puerto RS-232. [2.1]

Referencia en la tarjeta	Nombre	Descripción
D8	LED TX	Se ilumina cuando la transmisión en RS-232 está activa.
D9	LED RX	Se ilumina cuando la recepción en RS-232 está activa.

Tabla 2.9 LEDS de estado del puerto RS-232. [2.1]

2.3.7 Puerto de expansión GPIO 2x20.

La tarjeta cyclone V GX contiene dos puertos de 40 pines de expansión (GPIO) y un puerto de expansión Arduino Uno. Estos dos puertos GPIO comparten los encabezados de expansión de las entradas y salidas. Además, los puertos GPIO comparten las entradas y salidas con dos displays de 7 segmentos como se mencionó anteriormente. No se especifica cuantos dispositivos de expansión se pueden colocar sin embargo la programación del chip FPGA a un 100% es una ardua tarea y en caso de requerirse otro chip FPGA se puede colocar otra tarjeta.

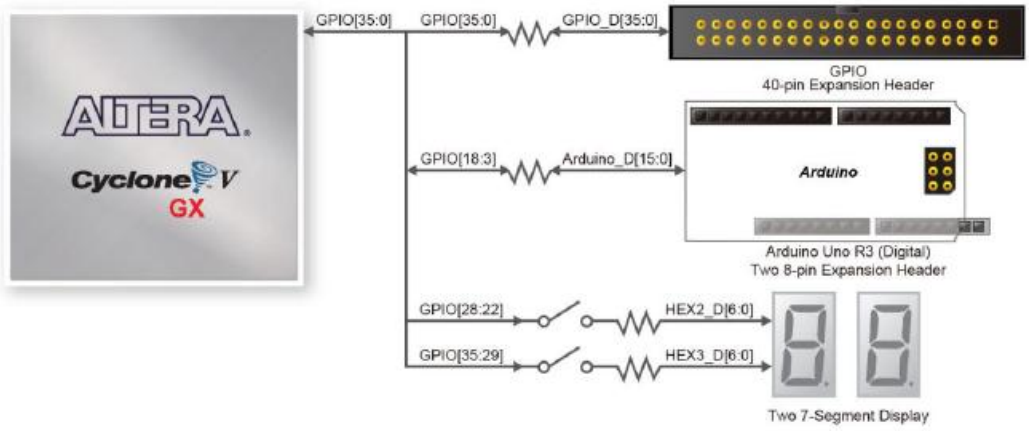


Figura 2.17 Diagrama esquemático interno entre chip cyclone V GX, puerto GPIO, Puerto arduino y displays de 7 segmentos. [2.1]

Puerto de expansión de 40 pines (GPIO).

El puerto de expansión de 40 pines se conecta directamente a 36 pines del chip cyclone V GX, y sus valores de corriente continua que proporciona son + 5V (VCC5), +3.3 V (VCC3P3), y dos pines GND. La figura 2.18 muestra la distribución de las entradas/salidas del conector GPIO. El consumo máximo de energía de la tarjeta secundaria que se conecta al puerto GPIO se muestra en la tabla 2.10.

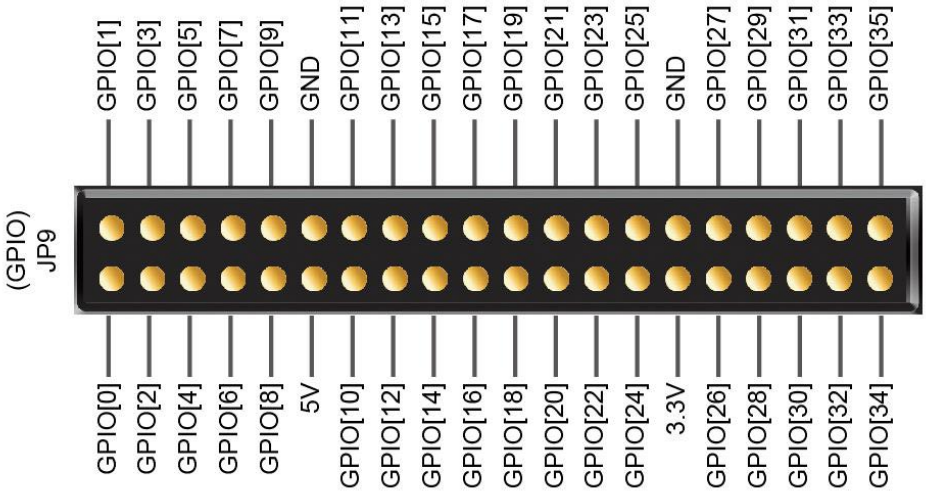


Figura 2.18. Arreglo de pines del puerto GPIO. [2.1]

Voltaje suministrado	Límite de máxima corriente
5 V	1.0 A
3.3 V	1.5 A

Tabla 2.10. Fuente de alimentación del puerto GPIO. [2.1]

Nombre de la señal	Bus que comparte	Descripción	E/S Estándar	Número de pin en chip cyclone V GX
GPIO0		GPIO DATA[0], Entrada dedicada de reloj	3.3 V	PIN_T21
GPIO1		GPIO DATA[1]	3.3 V	PIN_D26
GPIO2		GPIO DATA[2], Entrada dedicada de reloj	3.3 V	PIN_K25
GPIO3	Arduino_IO0	GPIO DATA[3], Arduino IO0	3.3 V	PIN_E26
GPIO4	Arduino_IO1	GPIO DATA[4], Arduino IO1	3.3 V	PIN_K26
GPIO5	Arduino_IO2	GPIO DATA[5], Arduino IO2	3.3 V	PIN_M26
GPIO6	Arduino_IO3	GPIO DATA[6], Arduino IO3	3.3 V	PIN_M21
GPIO7	Arduino_IO4	GPIO DATA[7], Arduino IO4	3.3 V	PIN_P20
GPIO8	Arduino_IO5	GPIO DATA[8], Arduino IO5	3.3 V	PIN_T22
GPIO9	Arduino_IO6	GPIO DATA[9], Arduino IO6	3.3 V	PIN_T19
GPIO10	Arduino_IO7	GPIO DATA[10], Arduino IO7	3.3 V	PIN_U19
GPIO11	Arduino_IO8	GPIO DATA[11], Arduino IO8	3.3 V	PIN_U22
GPIO12	Arduino_IO9	GPIO DATA[12], Arduino IO9	3.3 V	PIN_P8
GPIO13	Arduino_IO10	GPIO DATA[13], Arduino IO10	3.3 V	PIN_R8
GPIO14	Arduino_IO11	GPIO DATA[14], Arduino IO11	3.3 V	PIN_R9
GPIO15	Arduino_IO12	GPIO DATA[15], Arduino IO12	3.3 V	PIN_R10
GPIO16	Arduino_IO13	GPIO DATA[16], Arduino IO13, PLL clock output	3.3 V	PIN_F26
GPIO17		GPIO DATA[17]	3.3 V	PIN_Y9
GPIO18		GPIO DATA[18], PLL CLOCK OUTPUT	3.3 V	PIN_G26
GPIO19		GPIO DATA[19]	3.3 V	PIN_Y8
GPIO20		GPIO DATA[20]	3.3 V	PIN_AA7
GPIO21		GPIO DATA[21]	3.3 V	PIN_AA6
GPIO22	HEX2_D0	GPIO DATA[22]	3.3 V	PIN_AD7
GPIO23	HEX2_D1	GPIO DATA[23]	3.3 V	PIN_AD6
GPIO24	HEX2_D2	GPIO DATA[24]	3.3 V	PIN_U20
GPIO25	HEX2_D3	GPIO DATA[25]	3.3 V	PIN_V22
GPIO26	HEX2_D4	GPIO DATA[26]	3.3 V	PIN_V20
GPIO27	HEX2_D5	GPIO DATA[27]	3.3 V	PIN_W21
GPIO28	HEX2_D6	GPIO DATA[28]	3.3 V	PIN_W20
GPIO29	HEX3_D0	GPIO DATA[29]	3.3 V	PIN_Y24
GPIO30	HEX3_D1	GPIO DATA[30]	3.3 V	PIN_Y23
GPIO31	HEX3_D2	GPIO DATA[31]	3.3 V	PIN_AA23
GPIO32	HEX3_D3	GPIO DATA[32]	3.3 V	PIN_AA22
GPIO33	HEX3_D4	GPIO DATA[33]	3.3 V	PIN_AC24
GPIO34	HEX3_D5	GPIO DATA[34]	3.3 V	PIN_AC23
GPIO35	HEX3_D6	GPIO DATA[35]	3.3 V	PIN_AC22

Tabla 2.11 Asignación de pines para el puerto GPIO. [2.1]

2.4 Motor de corriente continua.

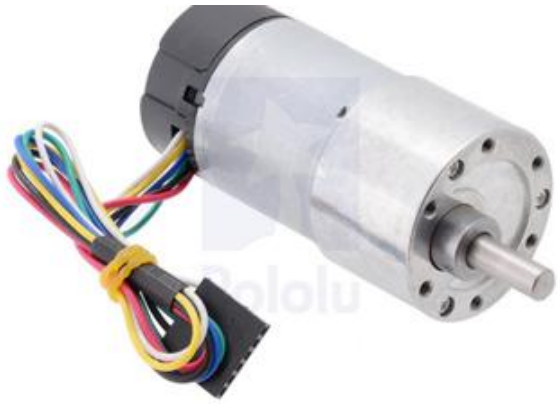


Figura 2.19 Motor de DC utilizado para el controlador PID. [2.2]

Características:

El tipo de motor usado es un motor DC de escobillas de imán permanente con caja reductora con una relación de 100:1, dicho motor tiene integrado un encoder de cuadratura el cual se explicara en la siguiente sección, dicho encoder provee 64 conteos por revolución (CPR) del eje secundario del motor lo que correspondería a un total de conteos de 64x100 = 6400 CPR para el eje principal.
El voltaje nominal del motor es de 12V aunque en general este tipo de motores pueden funcionar con voltajes alrededor del nominal aun con voltajes tan bajos como 1V (aunque este sea impráctico). Voltajes más altos del nominal afectan negativamente la vida del motor.
Dimensiones:

- Tamaño 37mm de diámetro x 72.5mm de Longitud
- Peso: 230g
- Diámetro del eje primario: 6mm

Especificaciones Generales:

Gear ratio:	100:1
Free-run speed @ 6V:	50 rpm
Free-run current @ 6V:	250 mA
Stall current @ 6V:	2500 mA
Stall torque @ 6V:	110 oz·in
Free-run speed @ 12V:	100 rpm
Free-run current @ 12V:	300 mA
Stall current @ 12V:	5000 mA
Stall torque @ 12V:	220 oz·in
Lead length:	11 in

Tabla 2.12. Especificaciones del motor. [2.2]

Definición de los parámetros:

- Run Spead: Velocidad del eje del motor sin carga.
- Stall current: es la máxima corriente obtenida cuando el motor está aplicando su máximo torque.
- Free Current: es la corriente obtenida cuando el motor gira sin carga a su máxima velocidad.
- Stall Torque: es el torque producido por el motor cuando la velocidad es cero, esto también incluye el torque de carga que causa que la velocidad del eje de salida sea cero.

2.5 Encoder acoplado al motor.

Tras la presentación del tipo de motor a usar, se vislumbra una de las ventajas de este. El motor trae incorporado un encoder de cuadratura de efecto Hall. Un encoder que utiliza sensores de efecto hall (son magnéticos) presenta ventajas contra condiciones ambientales que puedan afectar a los que funcionan con fotodiodos.
Como se mencionó anteriormente, el motor posee un eje reductor acoplado, la relación entre los radios de los ejes es de 100:1 Este parámetro importante se relaciona con otro ya mencionado: la resolución del encoder.

2.5.1 Relación de radios y Resolución.

Comenzando con la resolución, el encoder viene acoplado al eje secundario del motor (motor shaft) del cual si se toman los 2 canales en cuadratura se puede realizar una cuenta de 64 cambios de estado entre ambos canales. Para entender mejor este concepto, un ejemplo: se supone que se tiene un encoder de cuadratura que provee de 3 pulsos por canal por revolución del eje ver figura 2.20**,** se puede realizar un conteo de 12 cambios de estado usando los 2 canales.

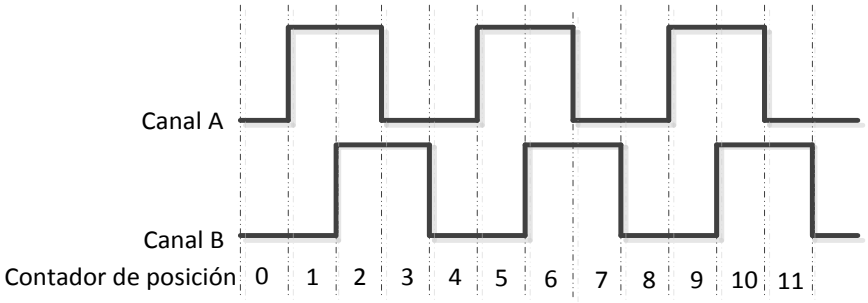


Figura 2.20. Pulsos por revolución de un encoder incremental. [2.2]

Este razonamiento dice que por cada canal del encoder acoplado al motor seleccionado se tiene 16 PPR de eje secundario por canal desfasados 90 grados. Una imagen real de las señales de cuadratura se muestra en la figura 2.22 y la configuración de los cables del encoder en la figura 2.21 y la tabla 2.13.

Color	Función
Rojo	Alimentation del motor
Negro	Alimentation del motor
Verde	Encoder GND
Azul	Encoder Vcc (3.5 – 20 V)
Amarillo	Encoder canal A
Blanco	Encoder canal B

Tabla 2.13. Clasificación de los cables del motor. [2.2]

Como características eléctricas es importante mencionar que el encoder requiere una alimentación separada del voltaje que alimenta al motor el cual puede estar entre 3.5 y 20 Volts y consumir un máximo de 10mA, dependiente de la alimentación, la señal de salida de los canales A y B debe alcanzar su valor máximo de 0 a VCC

Figura 2.11 Vista de planta del encoder acoplado al motor. [2.2]

Figura 2.22.. Vista en el osciloscopio de la salida del encoder. [2.2]

Siguiendo con el caso expuesto, el hecho de que el encoder este acoplado al eje secundario del motor ofrece una ventaja más para la resolución, ya que por cada 100 giros del eje secundario se tiene 1 del eje principal, también para cada 16 conteos del eje secundario (tomando en cuenta un canal) se tienen 16x100=1600 conteos en el eje principal para una vuelta de este y si se toman 2 canales se tendrían 64x100=6400 conteos lo que equivaldría a una resolución en modo 1x de:

$$Resolucion = \frac{360^0\ eje\ principal}{6400 conteos} = 0.056^0$$

Capítulo III: Software de programación de la tarjeta FPGA.

Introducción al capitulo III

Altera provee de software (una versión gratuita y una de paga) para llevar a cabo la programación de sus tarjetas, el software de programación es Quartus II el cual cuenta con todas las herramientas necesarias para el diseño de códigos en distintos lenguajes HDL y todas las facilidades de manejo del hardware de la tarjeta a la hora de la programación (por ejemplo el cambio de niveles de voltaje en las salidas, manejo de los pines de la tarjeta etc.). De igual forma con cada creación de un componente VHDL se simula (si se desea) el comportamiento de este mediante el programa Modelsim (versión gratuita y de paga) también proporcionado por Altera de modo que la compañía ofrece las herramientas necesarias para probar e implementar los diseños de sistemas digitales en las tarjetas.

Este capítulo es dedicado a la adquisición del software, su instalación, configuración y su uso en general de manera que provea los conceptos necesarios para llevar a cabo estas tareas de forma rápida para este y muchos otros diseños digitales.

3.1 Software de programación.

Luego de conocer las características de la tarjeta Cyclone V GX starter Kit de Altera, es importante familiarizarse con el software, se pretende sentar los primeros pasos para la creación de un proyecto en Quartus II web edition (versión free) y la creación de un proyecto de simulación en Modelsim, ambos de Altera usando el lenguaje de programación VHDL.

3.2 Descarga del software.

Para la descarga del software, el fabricante, en este caso Altera en su página web posee un centro de manejo de descargar gratuito y algunos otros con opción de adquirirlos con su licencia de pago, versiones disponibles de las herramientas para la programación de la tarjeta FPGA en el lenguaje VHDL, el nombre del software es: *Quartus II*, para poder obtener dicho software debe seguir el siguiente link: https://www.altera.com/downloads/download-center.html [en línea][última consulta 20/07/16].

Para realizar la descarga y obtener cualquier herramienta de software que el fabricante pone a disposición, se requiere llenar un formulario de registro, en el que se crea su cuenta de usuario con su respectiva contraseña, el link del sitio web es el siguiente: https://www.altera.com/mal-all/mal-signin.html [en línea] [última consulta 20/07/16].

Luego de cumplir los requisitos que supone crear su cuenta de usuario, se está a disposición de descargar la versión de *Quartus II* que el usuario considere necesario, es recomendable estar con las versiones más convenientes, en la actualidad la versión más reciente de *Quartus II* es la 16.0, sin embargo, al momento de realizar este trabajo, la última versión disponible que se encontraba era *Quartus* 15.0, y por cuestiones de rendimiento de la computadora disponible, se utiliza la versión de *Quartus II* 13.1, la figura 3.1 muestra la obtención de descarga del software, luego de elegir la versión del software (en este caso *Quartus II* 13.1) y observar si soporta programar el chip que contiene la tarjeta FPGA (en este caso Cyclone V) se debe hacer clic sobre la opción "Web Edition"

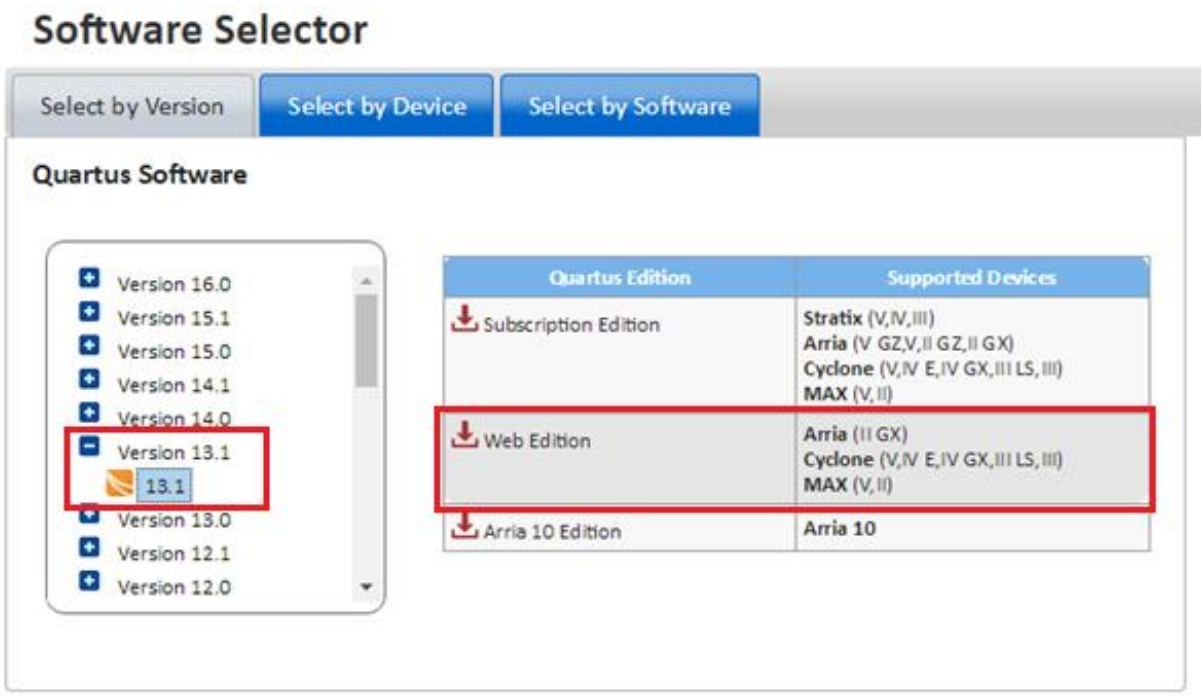


Figura 3.1 selección de la versión a descargar del software. [2.1]

Al hacer clic se redirige a otra parte de la web en donde se procede a la descarga del software, dicho software se encuentra disponible para sistemas operativos Windows y Linux, en este caso la descarga se realiza para sistema operativo Windows, es importante mencionar que el método de descarga del software se realiza mediante descarga directa.

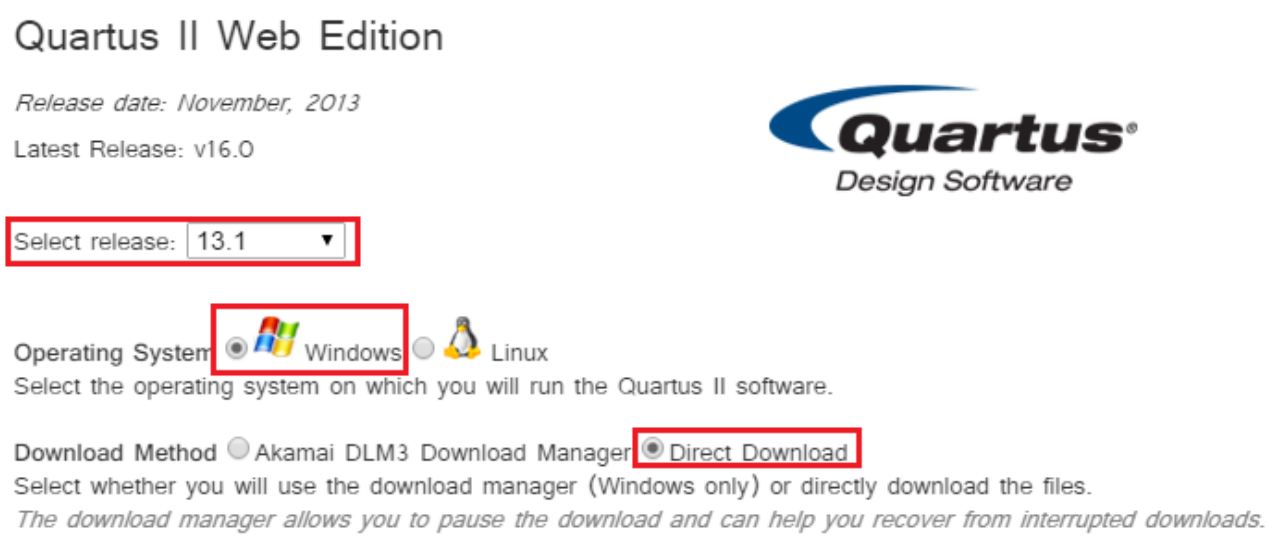


Figura 3.2 versión del software seleccionada para su previa descarga. [2.1]

Luego de esto, se tiene la opción de descargar por archivos individuales o el DVD completo, en esta ocasión se realiza mediante archivos individuales y se selecciona solamente los necesarios, para este caso, la tarjeta contiene su chip cyclone V y se selecciona el archivo "Cyclone V device support (includes all variations)" además de la descarga de *Quartus II*, como se muestra en la figura 3.3

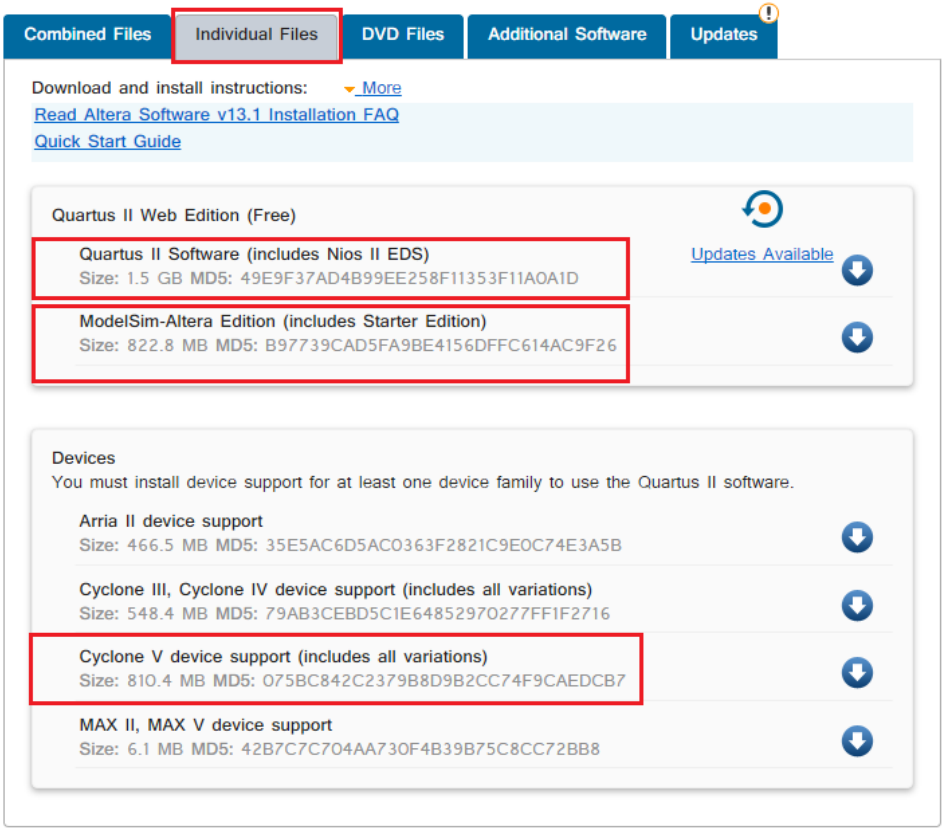


Figura 3.3 Descarga del software y archivos necesarios para su instalación [2.1].

Para poder colocar los archivos necesarios en el chip que contiene la tarjeta y así implementar su funcionamiento de acuerdo al programa realizado, se necesita la herramienta *"Quartus II Programmer and SignalTap II"* se procede a su descarga como lo indica la figura 3.4

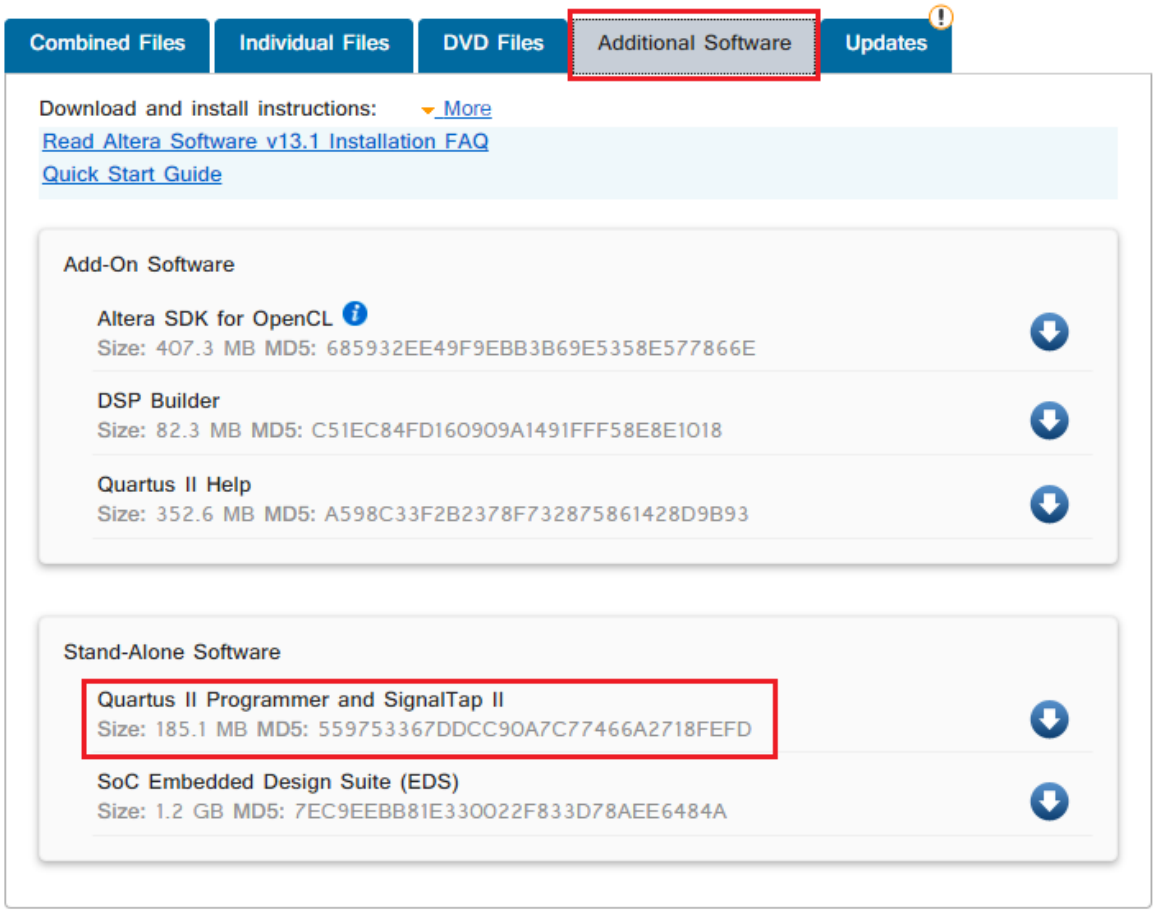


Figura 3.4 Selección para descarga de la herramienta Quartus II llamada Programmer and SignalTap II [2.1].

3.3 Instalación del software.

Una vez descargados todos los archivos, es importante que éstos se encuentren en la misma carpeta al momento de realizar la instalación del software *Quartus II*.

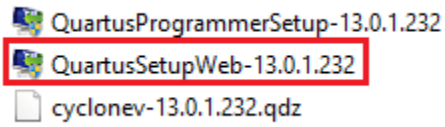


Figura 3.5 Archivos reunidos en una misma carpeta antes de la pre-instalación.

El fabricante (Altera) no específica requerimientos que la computadora debe cumplir para instalar *Quartus II,* más sin embargo se recomienda tener disponible una computadora que cumpla con las siguientes características:

- ✓ Microsoft Windows 7 o superior (de preferencia 64 bits).
- ✓ Procesador Dual Core 2.17 GHz como mínimo.
- ✓ 4 Gb de memoria RAM (recomendado).
- ✓ 15 Gb de espacio de disco duro.

Al compilar un código y simularlo la memoria RAM de la computadora es altamente demandada por lo cual se recomienda tener un valor alto de recurso, de esta forma se reduce el tiempo de ejecución.

Se ejecuta el archivo llamado "*QuartusSetupWeb-13.0.1.232"*, como se muestra en la figura 3.5, al seleccionarlo se ejecuta el proceso de instalación del software.

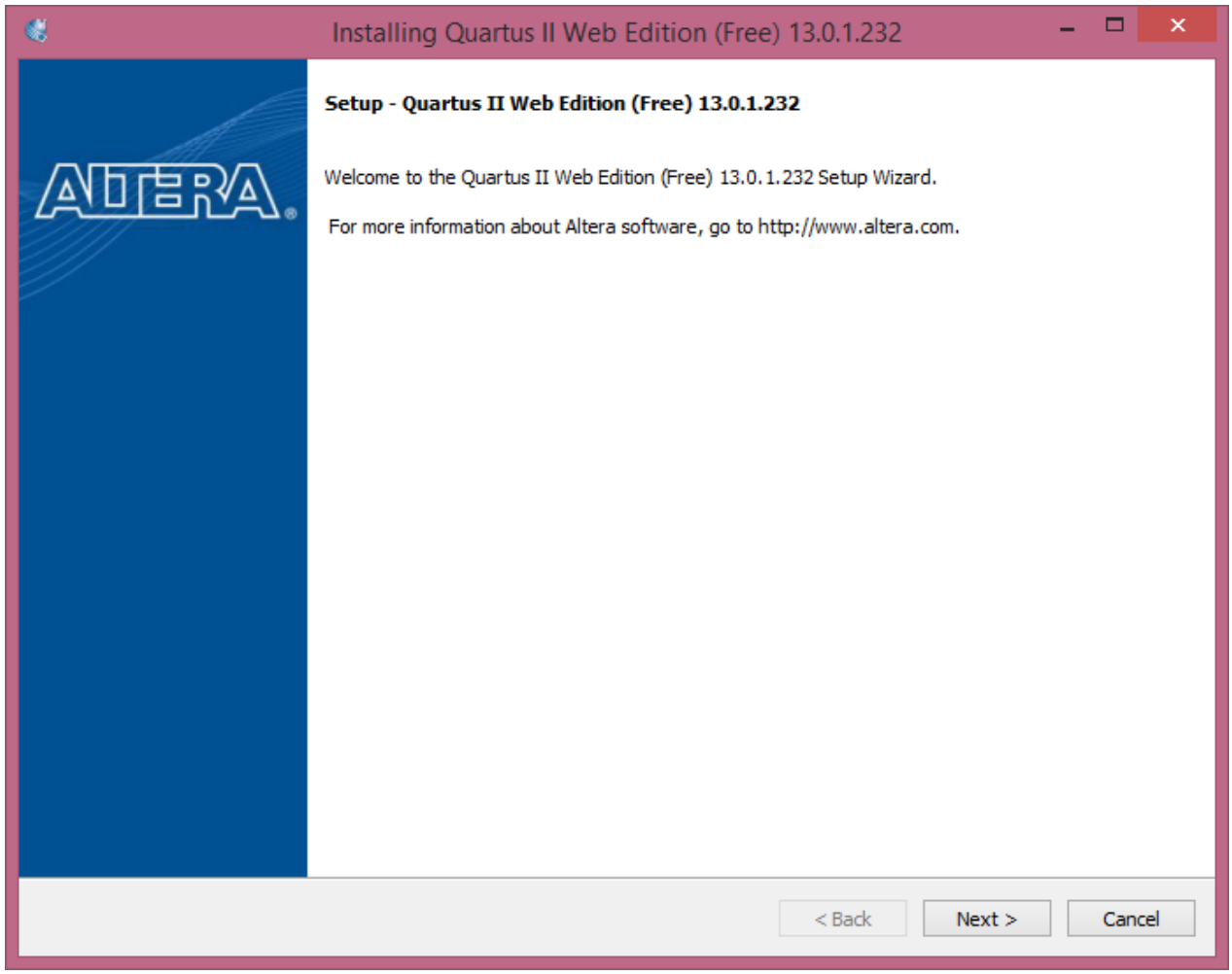


Figura 3.6 Pantalla inicial de instalación.

Luego de dar clic en "*Next"*, aparecen los términos y condiciones para el software *Quartus II* web edition, como se muestra en la figura 3.7

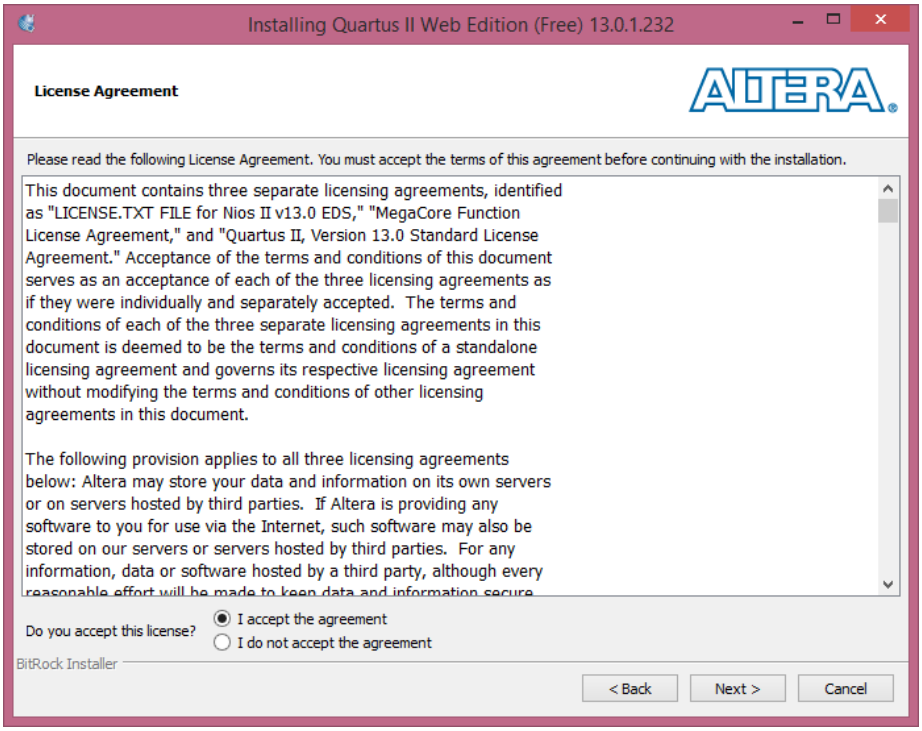


Figura 3.7 Términos y condiciones.

Al hacer clic en *"Next"* aparece la ventana de la figura 3.8 en donde se especifica la ruta de la instalación de *Quartus II*

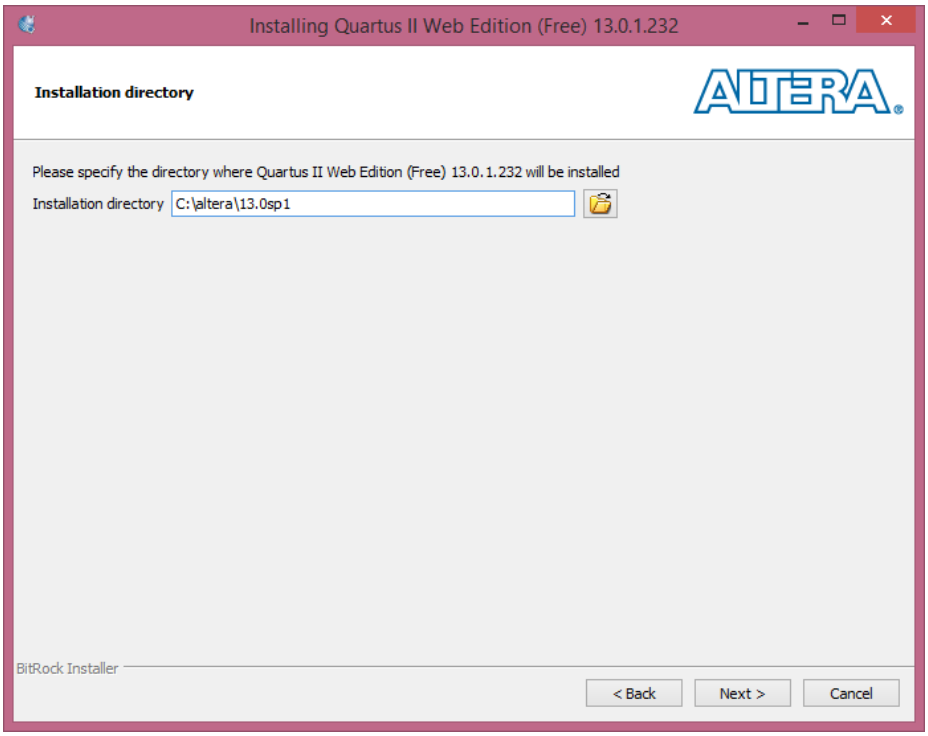


Figura 3.8 Dirección de instalación de Quartus II.

Como se mencionó los archivos descargados deben estar ubicados en la misma carpeta donde se encuentra el archivo *.exe* de *Quartus II* para que automáticamente se instalen, como lo muestra la figura 3.9

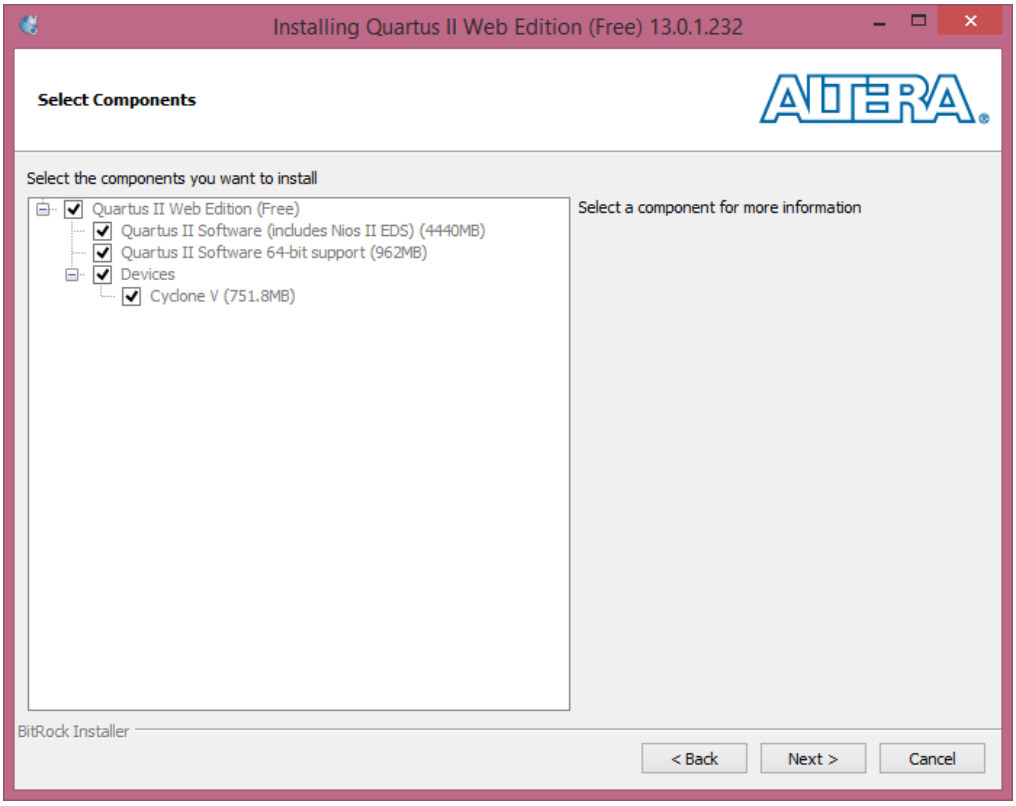


Figura 3.9 Componentes a instalar de Quartus II.

Al hacer clic en "*Next"* empezará el proceso de instalación del software, por los recursos que consume la instalación se recomienda cerrar todo programa que no sea de utilidad al momento de la instalación.
Al finalizar la instalación se debe instalar la aplicación *"Programmer"* que se utiliza para descargar en el chip cyclone el programa que el usuario ha diseñado para su previo funcionamiento.

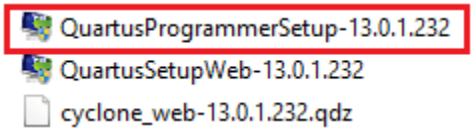


Figura 3.10 archivo de aplicación para la instalación de la herramienta programmer.

Al hacer clic en el archivo *"QuartusProgrammerSetup-13.0.1.232"* aparece la ventana de instalación como se muestra en la figura 3.11, luego de hacer clic en "*Next"* aparece la ventana que se muestra en la figura 3.12 y aceptar los términos y condiciones, se muestra la ventana en donde se muestra la ruta de instalación de la aplicación (figura 3.13). Es importante destacar que tiene que ser la misma ruta en la que se instaló *Quartus II,* luego inicia el proceso de instalación de la herramienta *programmer.*

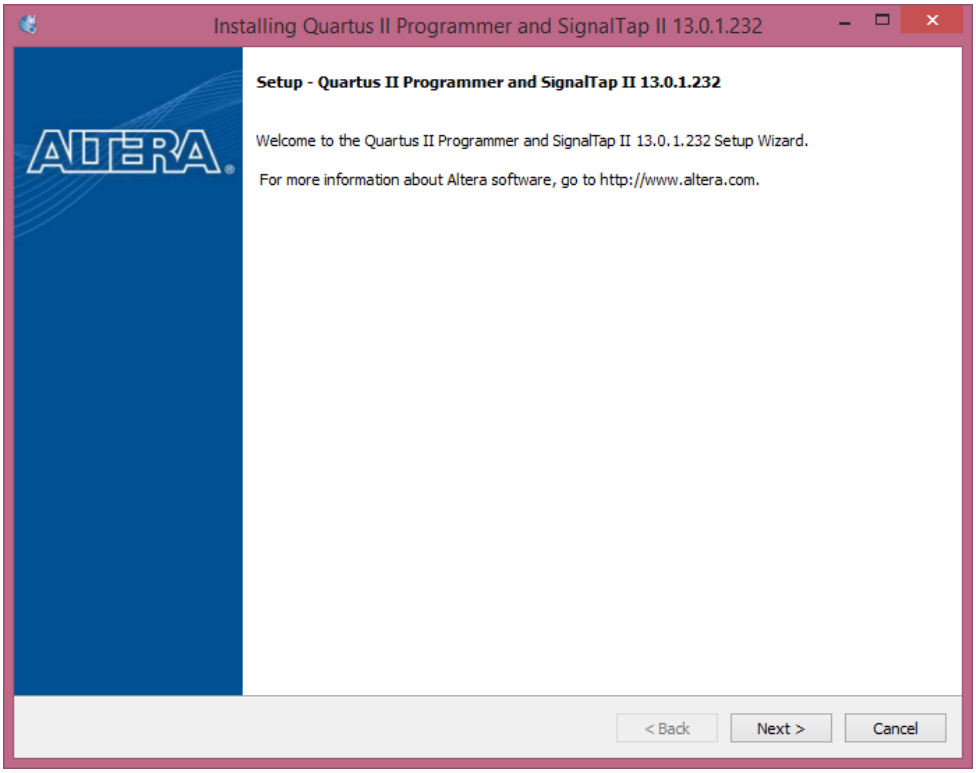


Figura 3.11 ventana de inicio de instalación de la herramienta programmer.

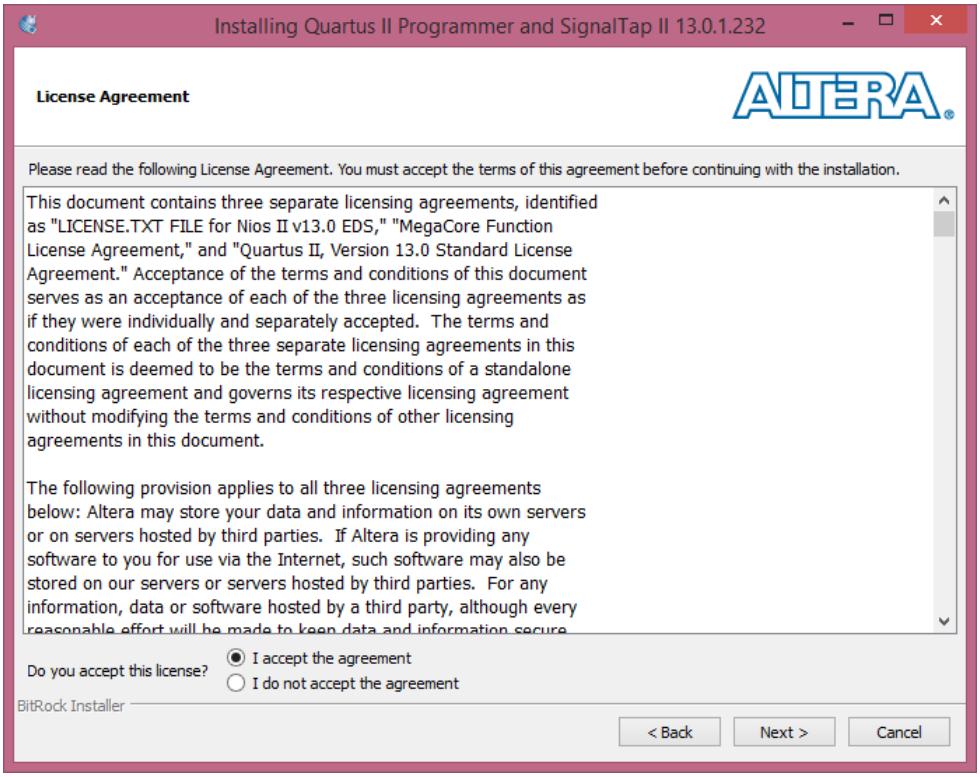


Figura 3.12 ventana de términos y condiciones.

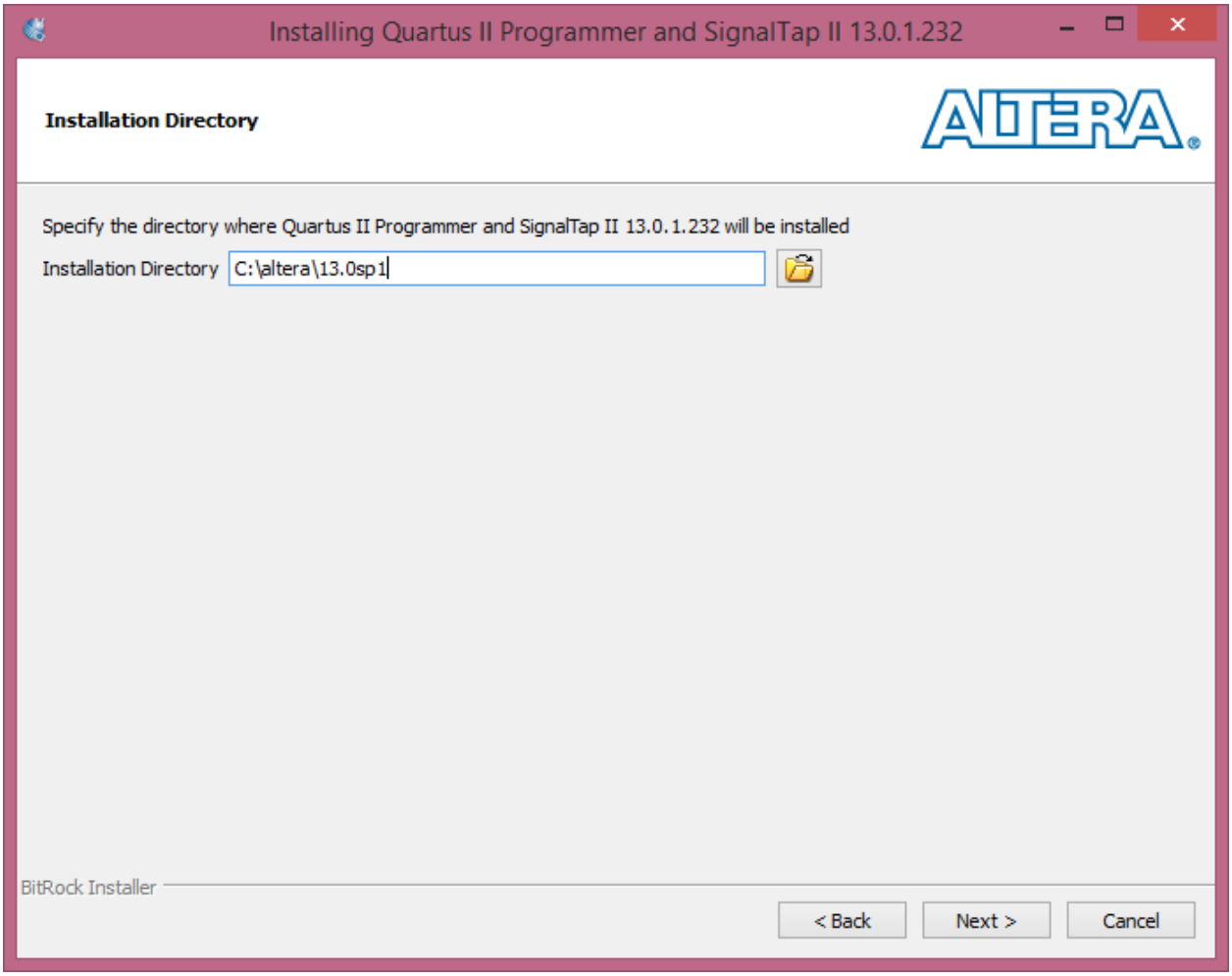


Figura 3.13 ruta de instalación de la herramienta programmer.

Al finalizar el proceso de instalación se está preparado para la creación de un proyecto en *Quartus II.*

3.4 Pasos para la creación de un proyecto en Quartus II.

Luego de realizar la instalación del software se procede a modo de ejemplo, la creación de un proyecto en *Quartus II*

1. Abrir el programa Quartus II de Altera. En esta ocasión se usara la edición WEB versión 13.1 ya que esta contiene los paquetes para el chip Cyclone V GX, Seleccionar la opción *"NEW PROJECT WIZARD"* que se encuentra dentro del menú de la opción *''FILE''*, como se muestra en la figura 3.14.
2. Dar clic *''SIGUIENTE"* en la ventana de introducción. A continuación se presenta la ventana en donde se especifica el directorio donde se guarda el proyecto, para este caso se debe usar la carpeta *"STUDENT"* que está en el siguiente directorio "C:/Altera/13.1/student", claro que podría ser otra ubicación. También se asigna un nombre al proyecto (PART1 para el caso) y este mismo nombre se copia en el nombre de la entidad de forma automática, esto se presenta en la figura 3.15.

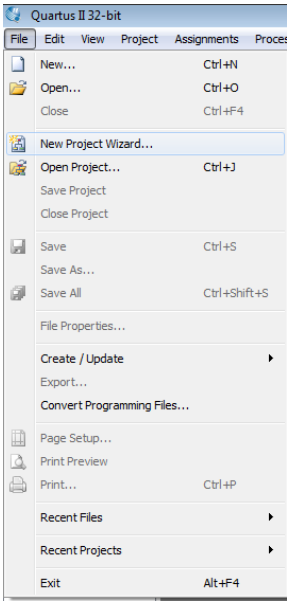


Figura 3.14 Crear un nuevo proyecto.

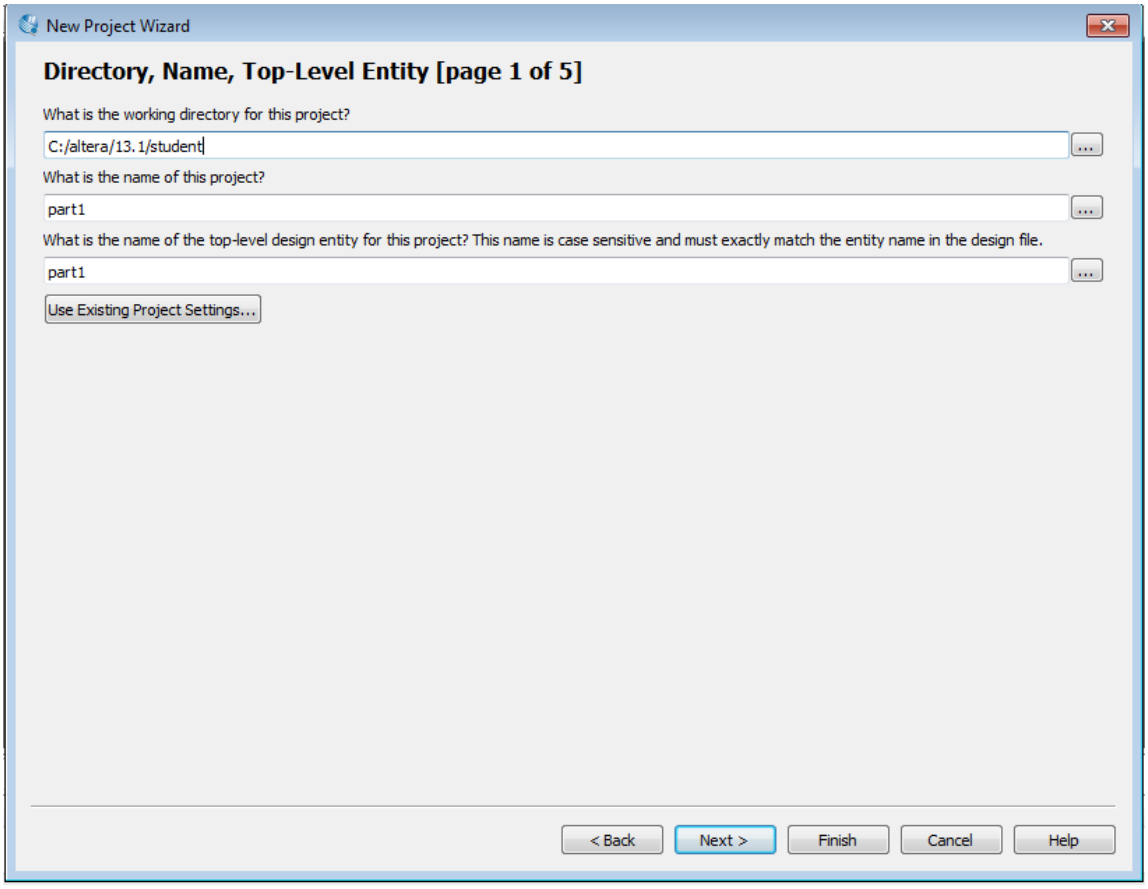


Figura 3.15 Directorio Y nombre del proyecto.

3. La ventana de *"ADD FILES"*, se agregan los archivos ya elaborados al código VHDL que se quiere crear, para el caso se parte de la creación del código desde cero por lo que no es necesario adjuntar algún archivo. Si por ejemplo se tiene un código VHDL que se desea modificar para que interactúe con el código principal, o un código que sirva de componente se puede adjuntar en esta ventana. Ver figura 3.16.

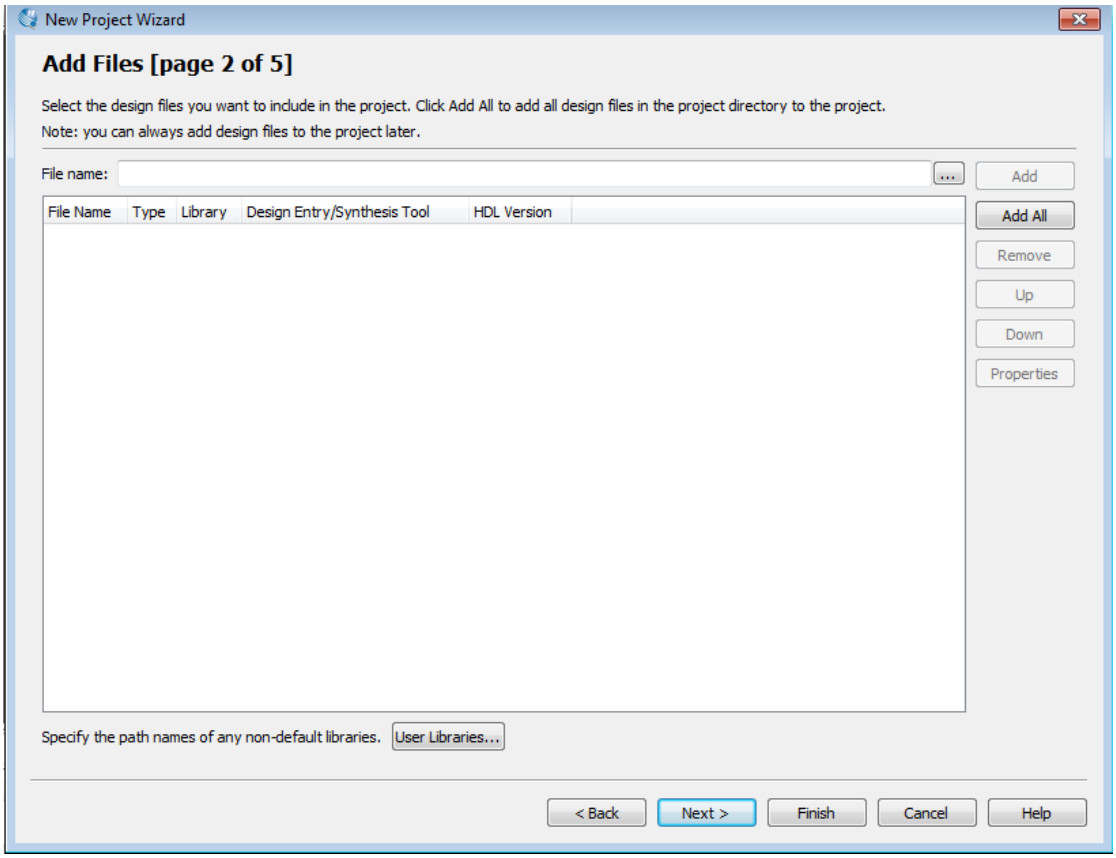


Figura 3.16 Ventana de ADD FILES.

4. La ventana de *"FAMILY AND DEVICE SETTINGS"* se debe seleccionar la familia del chip que posee la tarjeta FPGA. La tarjeta Cyclone V GX pertenece a la familia *CYCLONE V* y se muestra en el recuadro rojo de la figura 3.17 por lo que se debe seleccionar esta, el otro parámetro importante es seleccionar el chip en concreto con que cuenta la tarjeta el cual es: 5CGXFC5C6F27C7N. Ver recuadro amarillo de la figura 3.17.

5. La ventana EDA tool settings. Verificar la parte de *"SIMULATION"* debe estar seleccionado el programa Modelsim y VHDL ya que se usará el programa Modelsim para la simulación y se programará en VHDL. Ver recuadro rojo de la figura 3.18.

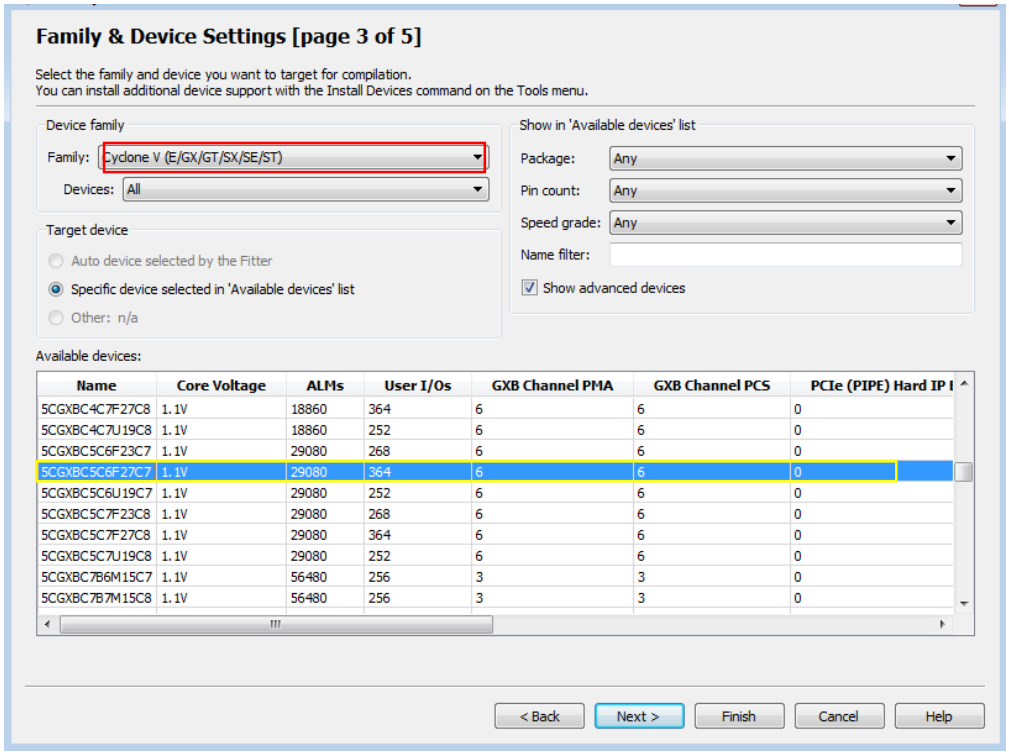


Figura 3.17. Selección de la familia y especificación del chip.

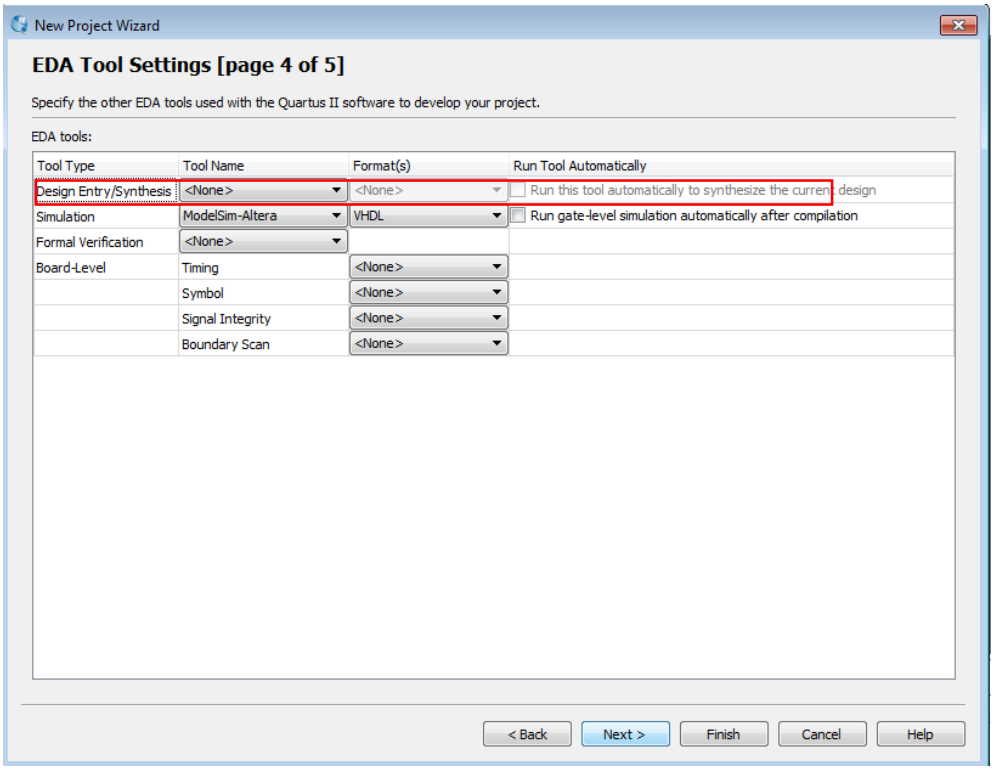


Figura 3.18 Selección del Programa de simulación y VHDL.

6. En la figura 3.19 se muestra la información que se ha estado completando en los pasos anteriores, Seleccionar la opción de *"FINISH"*

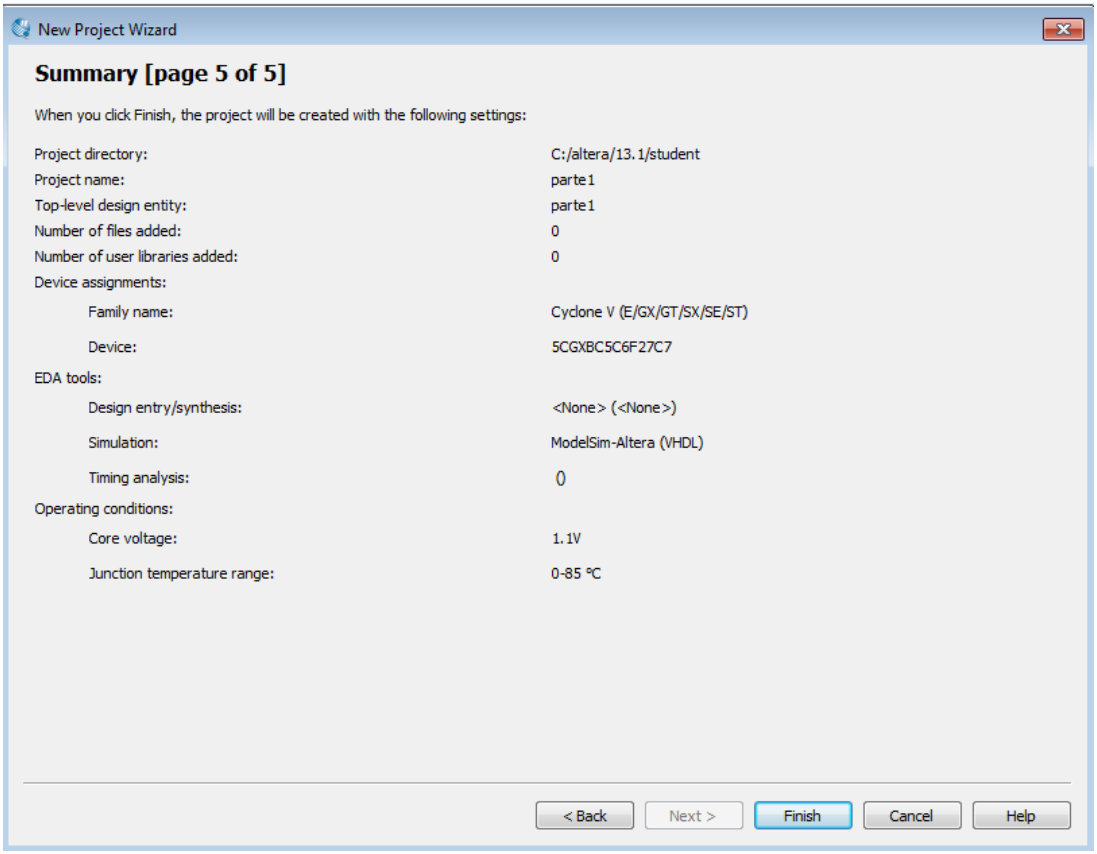


Figura 3.19 Datos del proyecto.

3.4.1 Ajustes para la simulación.

Es necesario realizar estos ajustes antes de realizar una compilación ya que con esto se evita la creación de ficheros múltiples para cada compilación (por defecto no se sobrescriben) además de establecer la ruta en la que se desarrolla el proyecto ya que el programa siempre dirige los archivos a la ruta por defecto (la carpeta student dentro del directorio Altera).
Por lo que en los siguientes pasos se realizan los ajustes para tener un solo archivo (.vho ó .vhd) necesarios para la simulación en Modelsim y establecer la ruta correcta para su uso.

Pasos:

1. Se debe verificar la dirección donde se guardan los datos del proyecto correspondiente a la simulación, para esta situación el siguiente paso es seleccionar en el menú *"ASSIGMENTS"*, la opción *"SETTINGS"*. Ver figura 3.20.

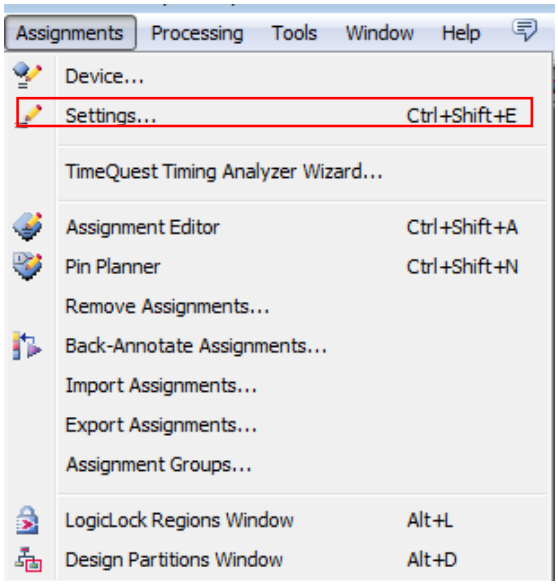


Figura 3.20 Seleccionar la opción Settings

2. Dentro de la siguiente ventana se selecciona la categoría *"EDA TOOL SETTINGS--SIMULATION"* y la opción *"MORE EDA NETLIST WRITER SETTINGS"*.

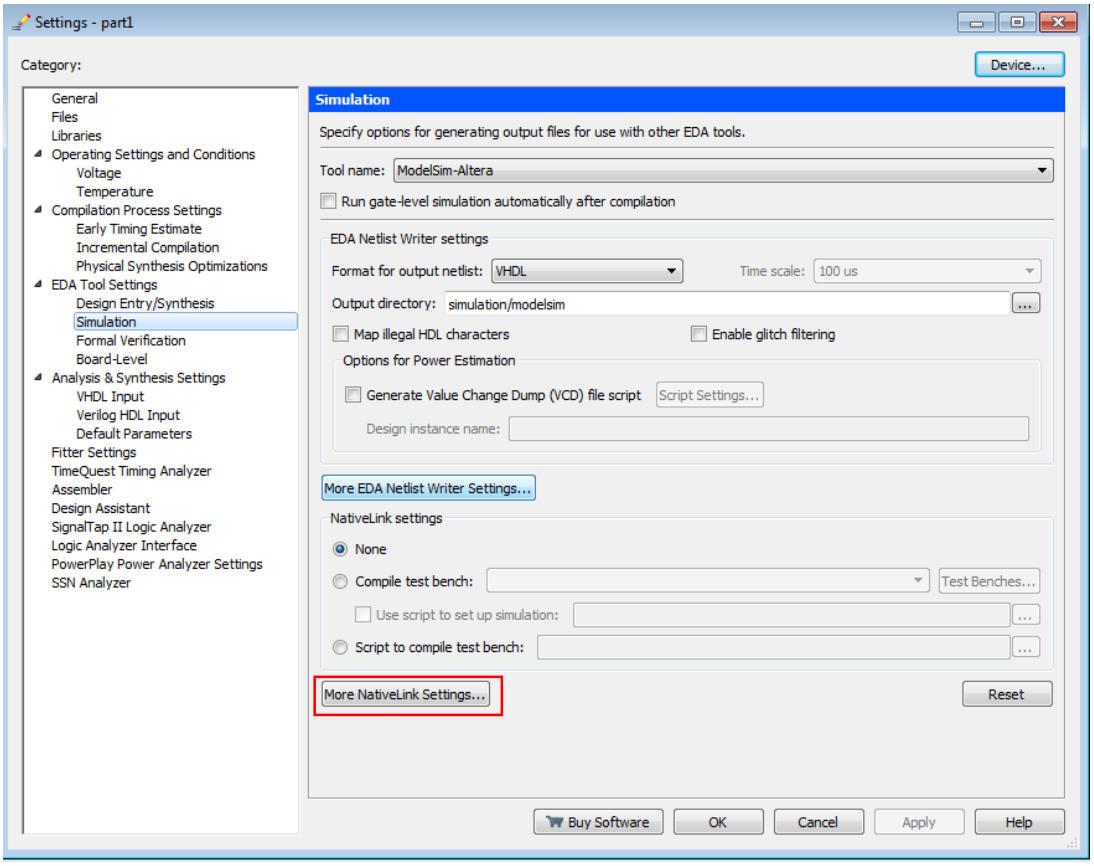


Figura 3.21 Selección de la opción More EDA netlist writer setting

3. Verificar que la dirección que se muestra en la figura 3.22. Sea la misma donde se ha guardado el proyecto, además de poner en *"ON"* la opción que está marcada en azul *"GENERATE NETLIST FOR FUNCTIONAL SIMULATION ONLY"* para crear archivos de simulación solo con compilaciones exitosas de este modo se tendrá un solo archivo para la simulación. dar clic en OK.

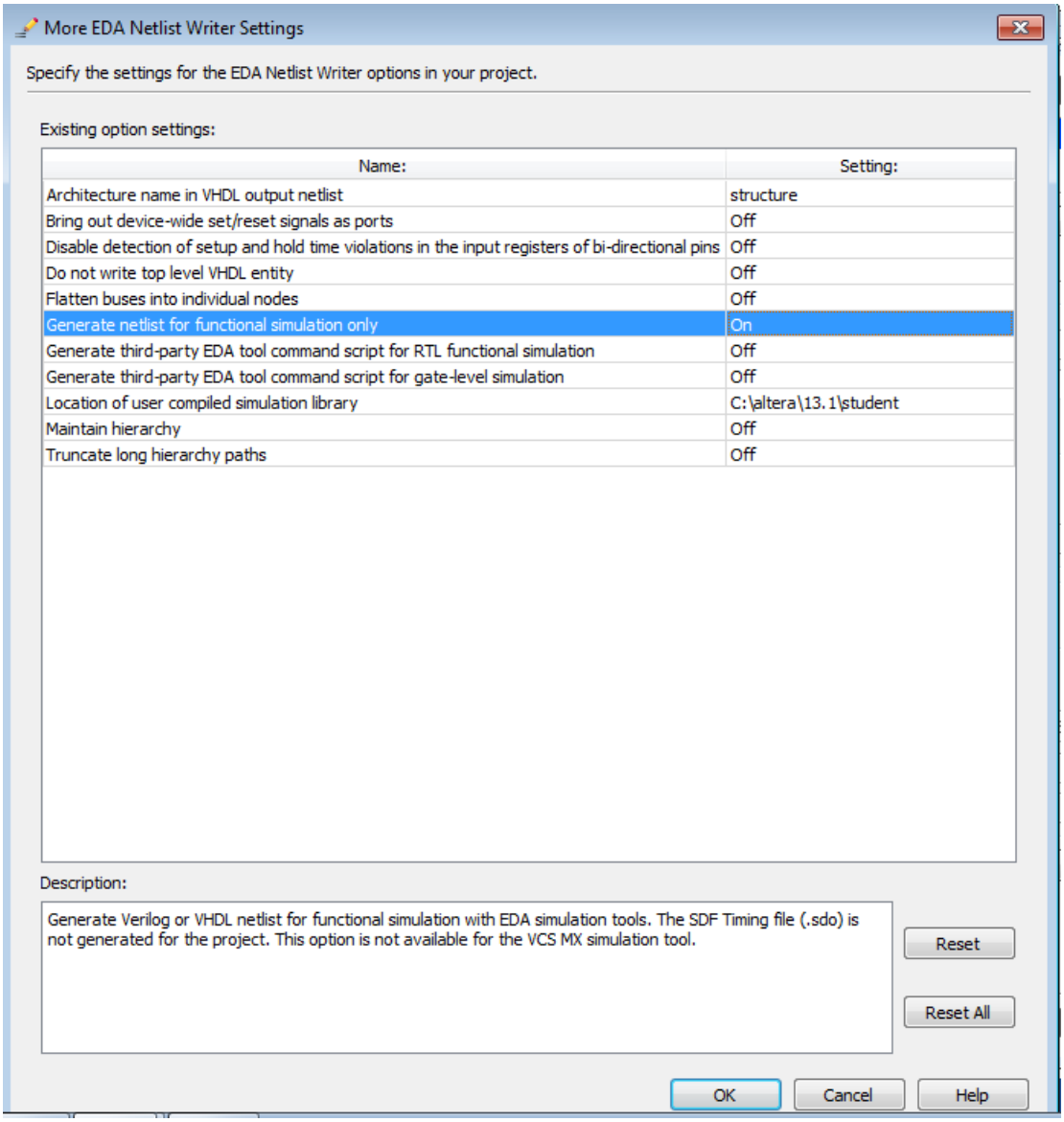


Figura 3.22. Ajustes para la simulación del código VHDL

4. Entrar a la opción *"MORE NATIVELINK SETTINGS"* y verificar que la dirección del proyecto aparece seleccionada en la casilla *"Location of user compiled simulation library"*, finalizar dando *"ok"*.

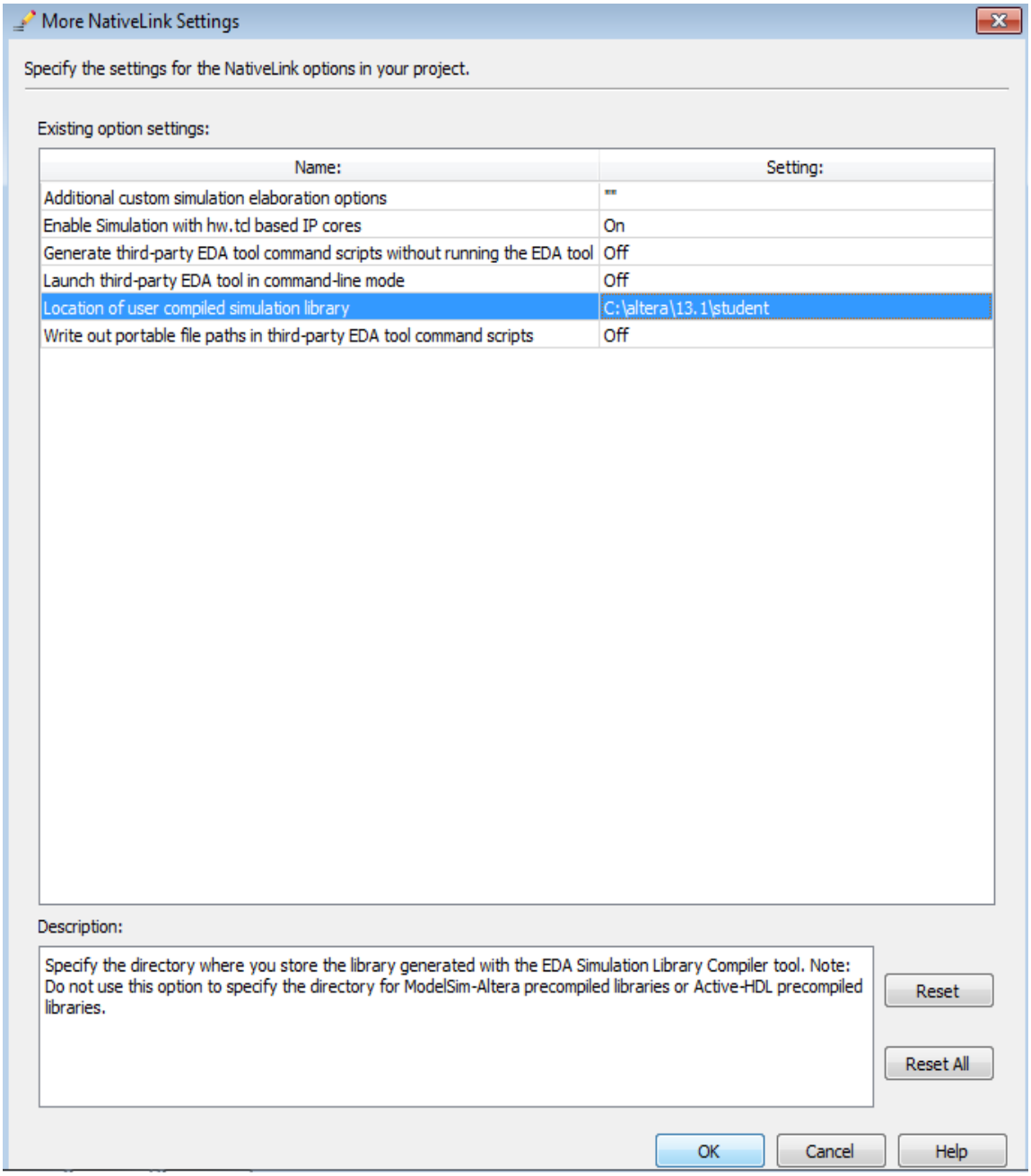


Figura 3.23 Ajustes finales para la simulación.

3.4.2 *Creación del código VHDL, compilación.*

1. Continuando el procedimiento, se debe escribir el código1 VHDL para su posterior simulación y asignación de pines, para esto se debe ir al menú *"FILE"* y seleccionar la opción *"NEW"*. Con lo que aparece la ventana que se muestra en la figura 3.24. En la que se debe seleccionar VHDL como archivo de diseño.

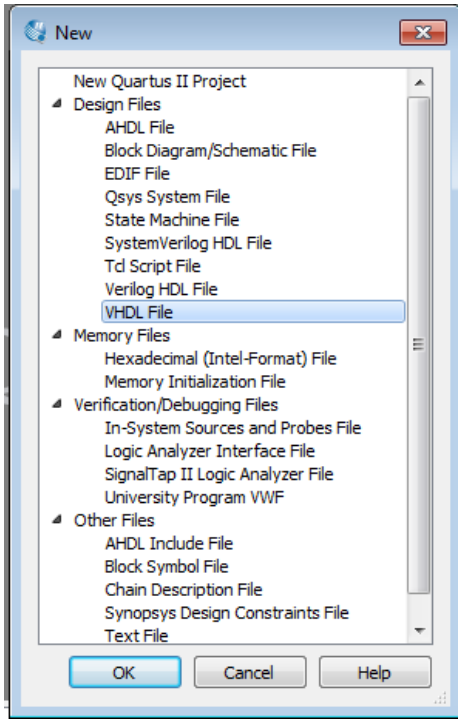


Figura 3.24 Selección del lenguaje VHDL como archivo de diseño.

2. Escribir el código VHDL, guardar el código y luego hacer clic en el icono correspondiente a *"start compilation"* que se muestra en la figura 3.25. Adicionalmente dicha opción se encuentra en el menú *"PROCESSING"* o *"CTRL+L"*.

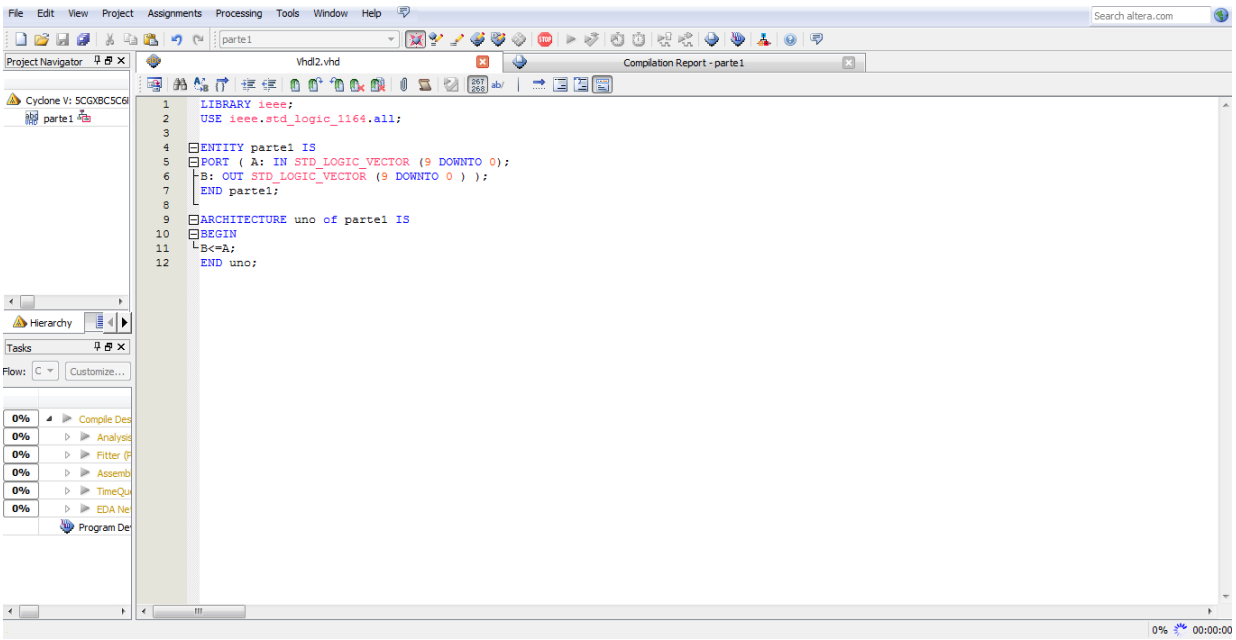


Figura 3.25 Escritura y compilación del código VHDL.

La compilación muestra una ventana de resultados, de ser exitosa debe observarse como en la figura 3.26. En ocasiones la falta de direccionamiento sobre los pines de la tarjeta suele provocar algunas alertas “warning”, estas se resuelven al asignar los pines correspondientes (ver sección 3.6)

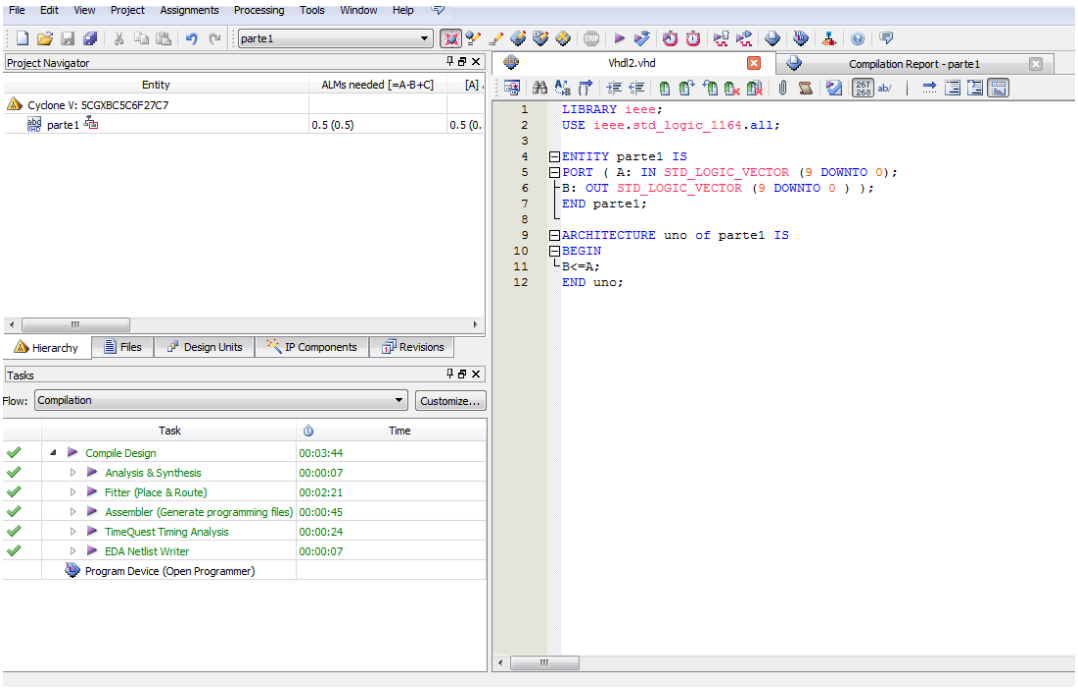


Figura 3.26 Resultado de la compilación.

3.5 *Simulación.*

La respuesta de como probar el diseño se encuentra en la simulación, para esto se necesita del programa Modelsim de Altera, los ajustes previos ya fueron hechos así que se describen los pasos para realizarlo.

1. Abrir el programa Modelsim de Altera edición starter 10.1d, del menú ‘’FILE’’ seleccionar la opción ‘’NEW’’ y luego ‘’PROJECT’’, como lo muestra la figura 3.27.

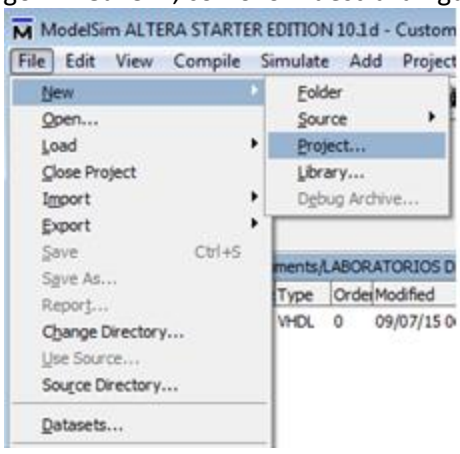


Figura 3.27. Creando un nuevo proyecto de simulación.

2. En la siguiente ventana especificar como dirección del proyecto de simulación la misma dirección que se creó para el proyecto en Quartus II, asignar un nombre de proyecto que puede ser cualquiera. Terminar con Ok

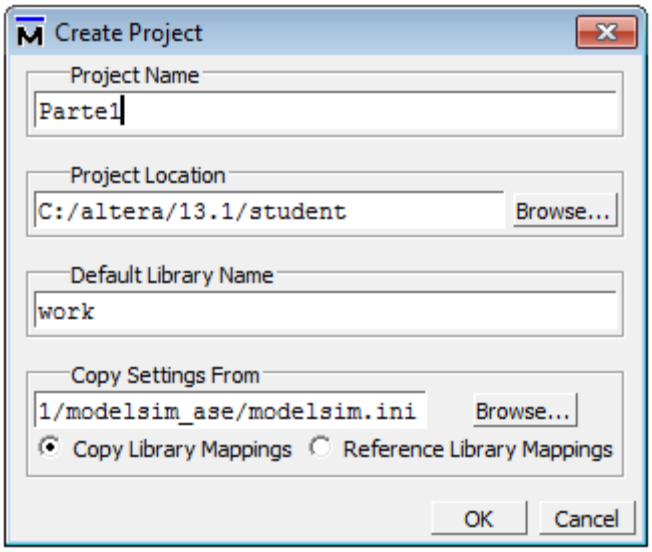


Figura 3.28. Nombre y localización del proyecto.

3. Seleccionar en la siguiente ventana (figura 3.29) *Add existing file* y buscar en la carpeta que se ha creado para el proyecto (figuras 3.30, 3.31) la carpeta con nombre "simulation" que contendrá la carpeta de Modelsim en la que estará el archivo part1.vho necesario para la simulación, otra opción es tomar el archivo .vhd que se ha generado tras una compilación exitosa, este archivo es el que contiene el código vhdl que se ha escrito y está en el directorio que se ha creado para el proyecto.

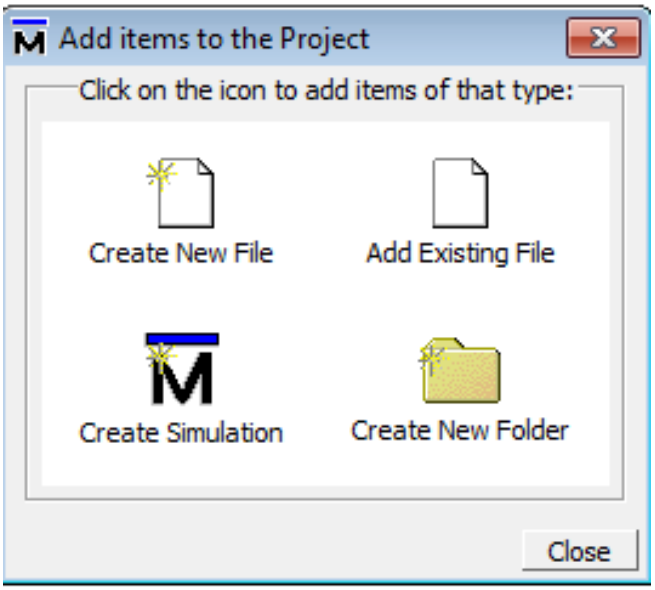


Figura 3.29. Selección de un archivo existente

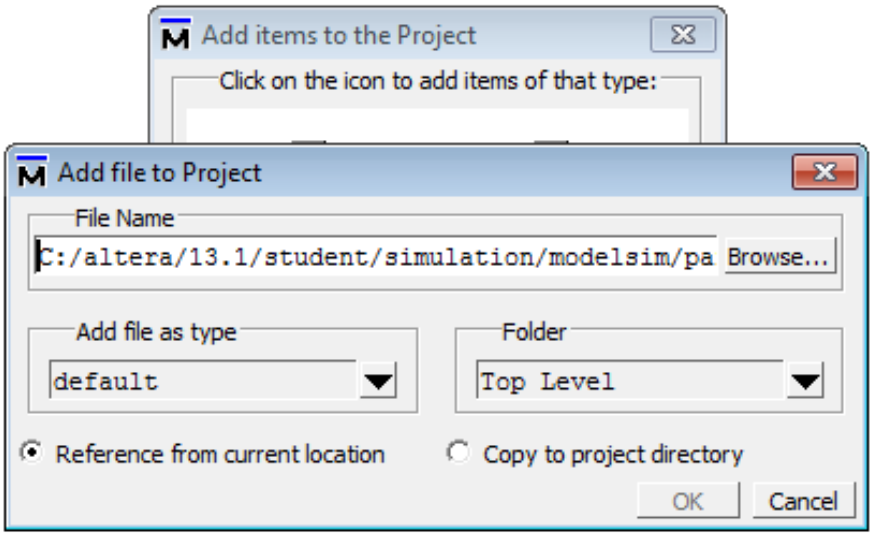


Figura 3.30. Búsqueda del archivo.

4. Seleccionar el archivo cargado con clic derecho y clic en la opción *"COMPILE SELECTED"* ver figura 3.31.

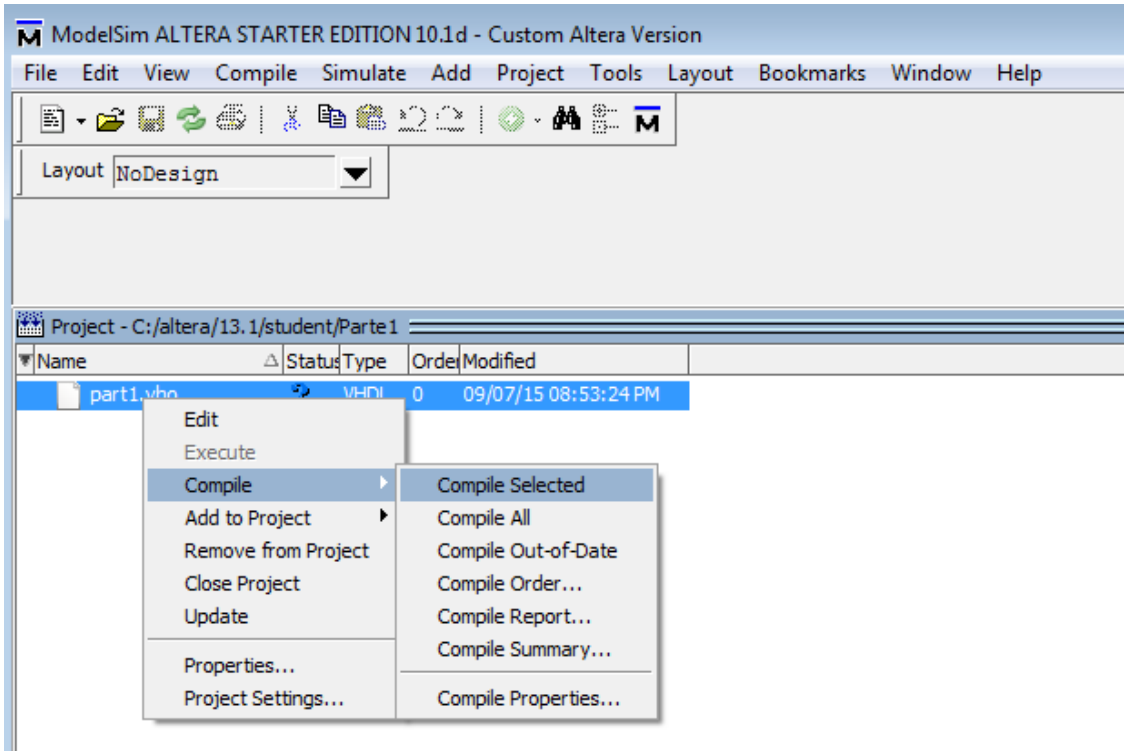


Figura 3.31. Compilación del archivo part1.vho.

Luego de esto en lugar de un signo de interrogación en la columna *"STATUS"* aparecerá un cheque.

El siguiente paso es realizar la simulación, para lo que se dará clic en el menú *"SIMULATE"* y luego en *"START SIMULATION"*.

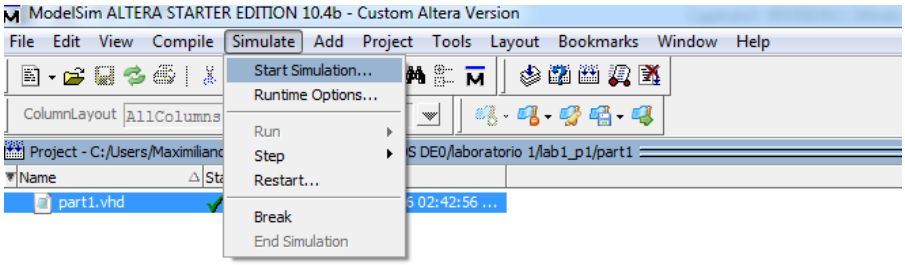


Figura 3.32 Menú para simulacion

La librería asignada para la simulación fue la "WORK" esta es la librería por defecto y es preferible no modificarla. En esta librería se abre el sub menú y se encuentra el archivo de tipo entidad part1 ver figura 3.33, se procede seleccionándolo y dando "*Ok*".

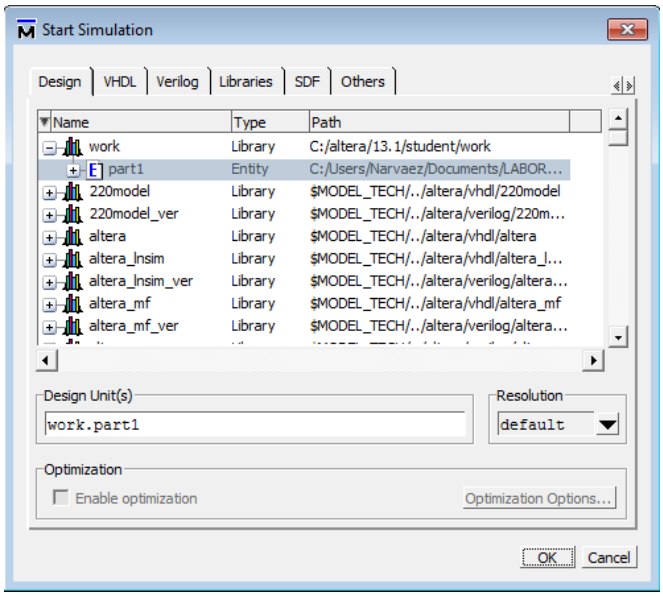


Figura 3.33 Selección del archivo part1

5. En la ventana emergente se selecciona la subventana *"OBJECTS"* en ella se encuentran todas las señales que se han usado en el código VHDL, para su análisis en el tiempo se debe seleccionar la señal que se quiere estudiar, hacer clic derecho sobre ella y seleccionar *"ADD TO"*, *"WAVE"*, *"SELECTED SIGNALS"* (ver figura 3.34), después de seleccionar todas las señales de interés seleccionar la subventana *"WAVE"*.

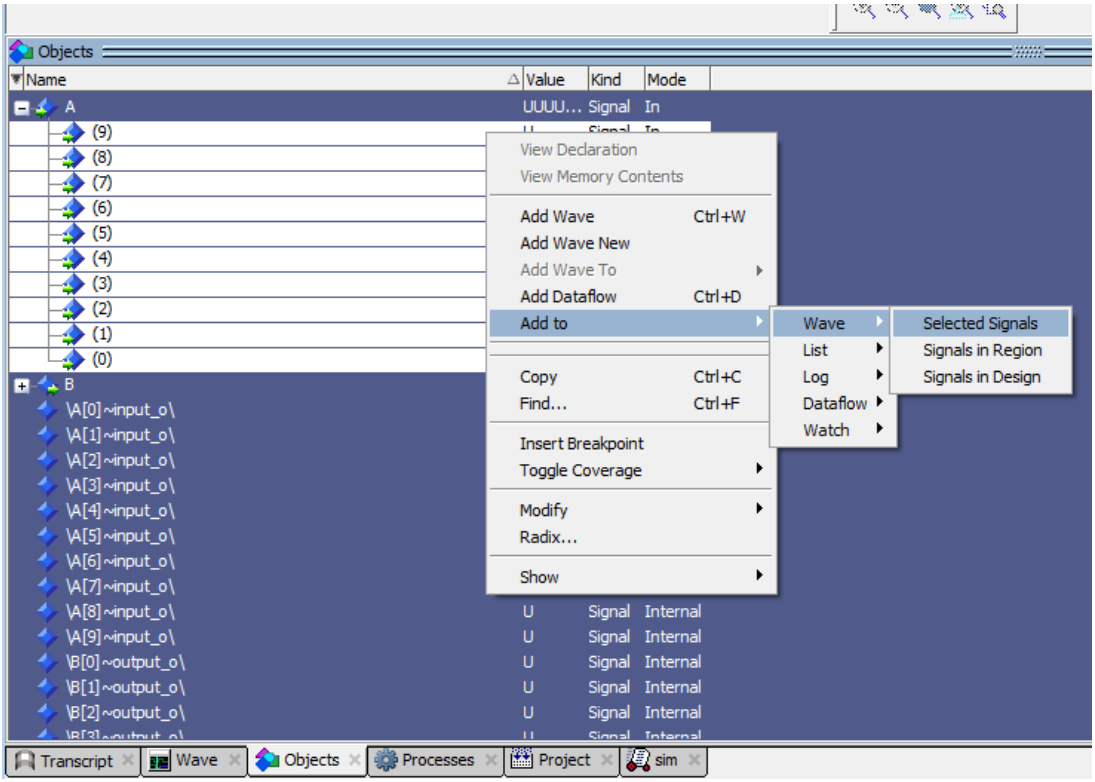


Figura 3.34. Agregar señales para visualizar

6. En la subventana de wave una forma práctica de ver el comportamiento en el tiempo de las señales es asignar a las señales de entrada la forma de pulsos periódicos como un “reloj”, cada entrada con diferente periodo para notar el comportamiento para varias situaciones de entrada en la salida (ver figura 3.35 y 3.36)

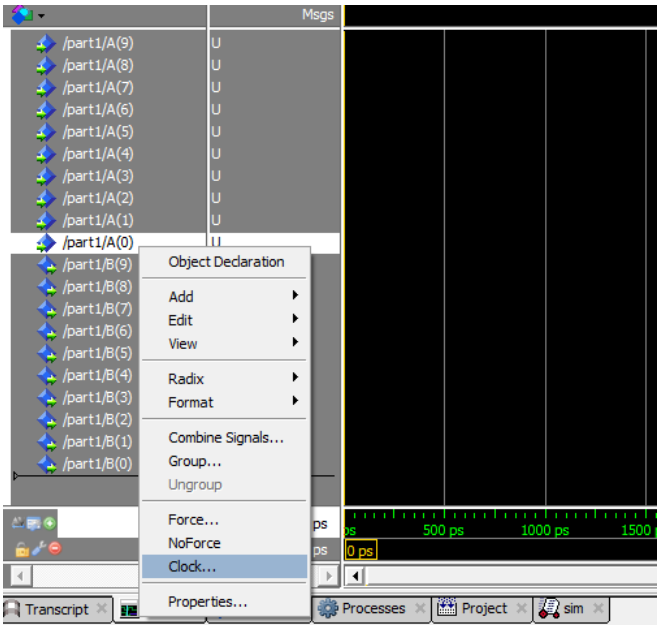


Figura 3.35. Seleccionar la opción Clock después de dar clic derecho sobre una señal.

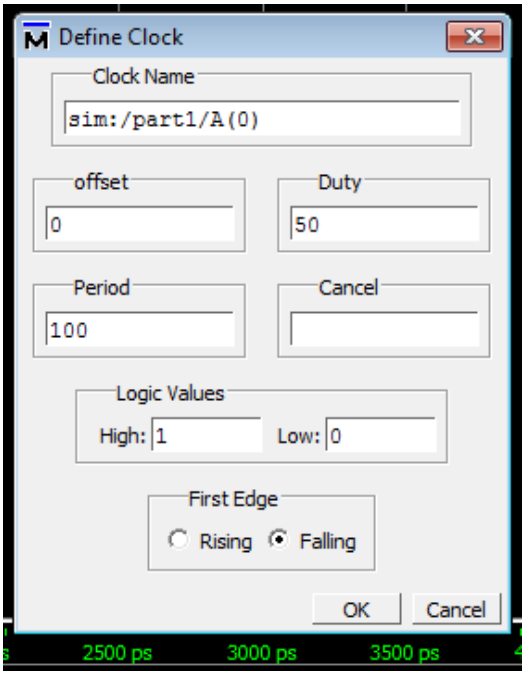


Figura 3.36. Seleccionar un periodo de 100ns y falling.

Subsecuentemente, para las demás señales el periodo aumenta (de 100 en 100 para esta prueba) por lo que A(1) tendrá un reloj con periodo de 200 y en falling, A(3) de 300, etc.

7. Después de asignar valores a todas las entradas, seleccionar la opción *"RUN"* que se encuentra en el menú *"SIMULATION"*, o en su defecto la tecla *"F9"*, se muestran las señales de entrada y las salidas. Por la naturaleza del diseño las salidas serán igual que las entradas ver figura 3.37.

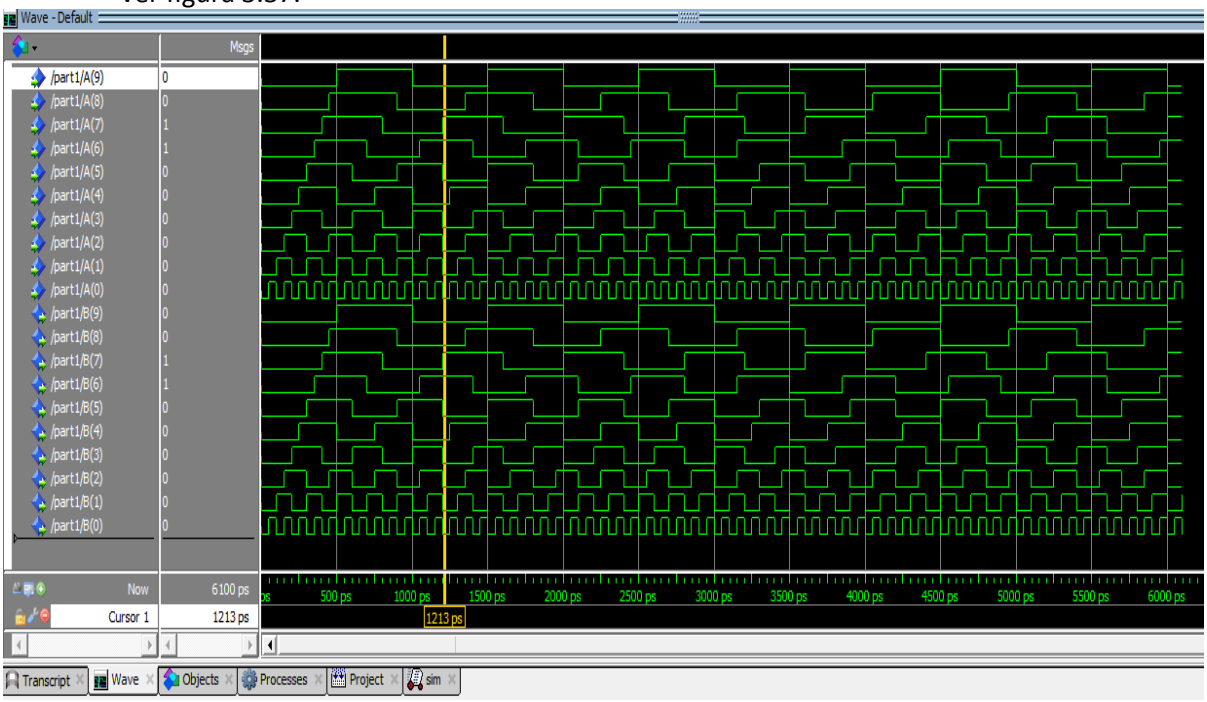


Figura 3.37. Resultado de la simulación.

3.6 Asignación de pines.

La asignación de pines se hace mediante la opción *"PIN PLANNER"* que se encuentra en el menú *"ASSIGMENTS"* del programa Quartus II. Después de elegirla se abrirá una ventana (ver figura 3.38), en la que se debe ubicar el pin del chip correspondiente a cada entrada y salida. Por ejemplo como ya se mencionó se utilizan los pines designados para los Switchs y Leds, dichos pines deben ser escritos en el cuadro que corresponde a *"LOCATION"* en la ventana de pin planner, así para la entrada A(9) en el código se debe asignar el correspondiente SW que para el caso es el (SW9) y este corresponde pin_D2 en la tarjeta Cyclone V GX (basta con escribir D2).

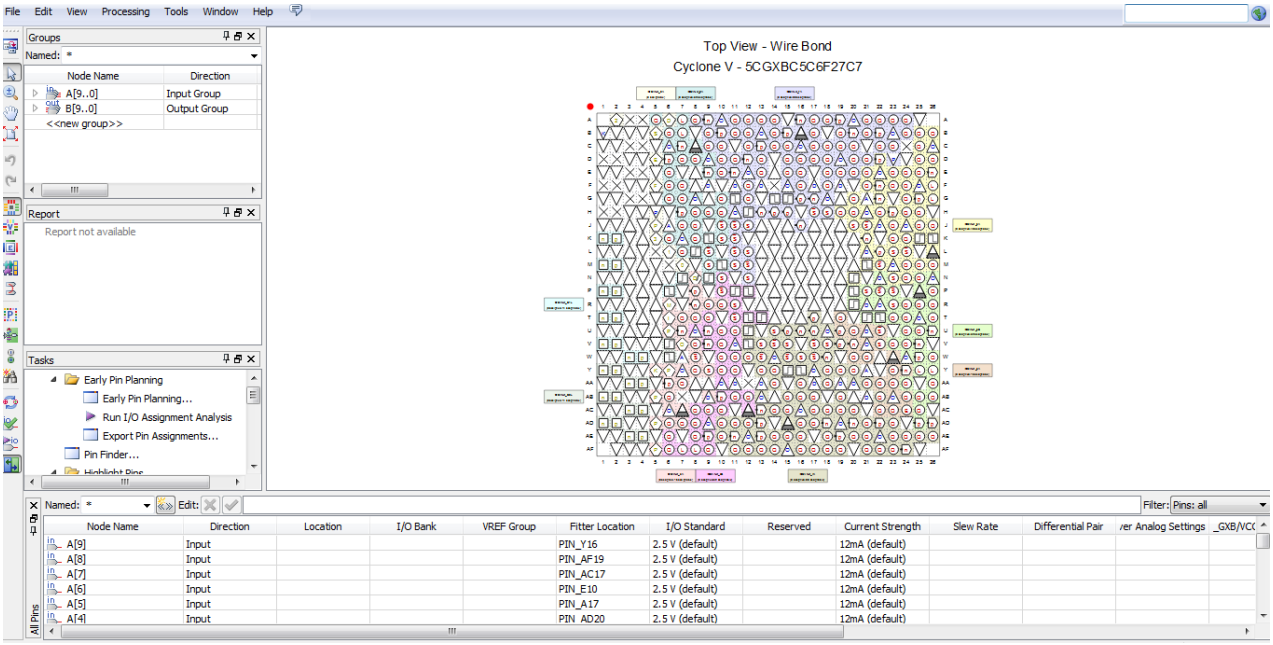


Figura 3.38. Asignación de pines.

Luego de la asignación de pines se debe cerrar la ventana de pin planner, recompilar el código y se puede proceder a la implementación en la tarjeta Cyclone V FPGA.

3.7 Implementar el diseño en la tarjeta cyclone V GX Starter.

Es importante que el driver USB-BLASTER esté instalado en la computadora para que pueda reconocer la tarjeta. Otro aspecto necesario es tener conectada la tarjeta en el conector USB de la computadora, encenderla con su cargador conectado y dl switch de programación en RUN.
El proceso de implementación en la tarjeta es el ultimo a realizar ya que se debe saber que el diseño funciona como se desea mediante la simulación, la asignación de pines es necesaria de otra manera se mostrara un error. Una vez se han realizado los pasos anteriores se procede desde el programa Quartus II.

1. Ir al menú "TOOLS" y seleccionar *"PROGRAMMER"*, ver figura 3.39.

2. En este paso se debe seleccionar en *"HARDWARE SETUP"* el hardware USB-BLASTER (Ver figura 3.40). Luego de esto se verifica el dispositivo y el "MODE" es JTAG, con esto se procede seleccionando la opción *"START"*, si el proceso ha sido exitoso la barra de *"PROGRESS"* resalta de color verde.

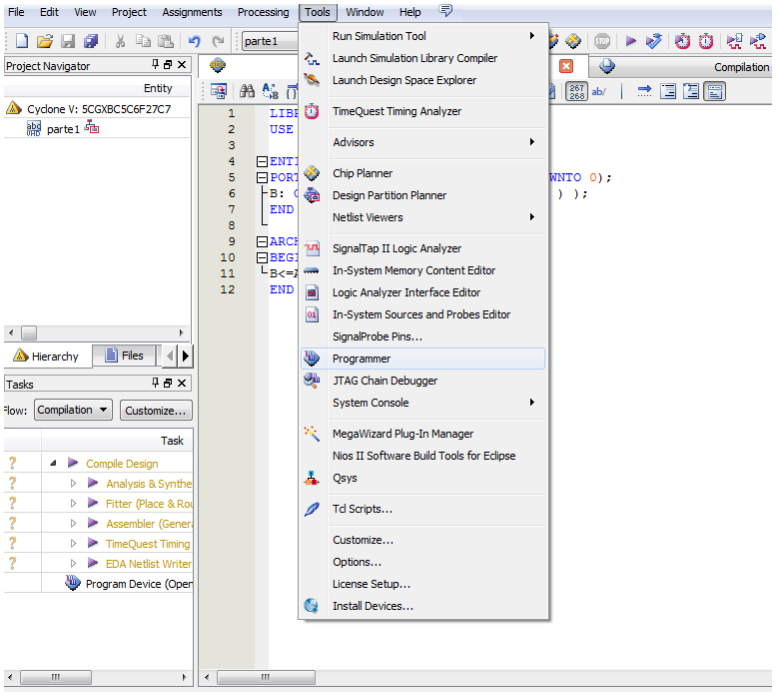


Figura 3.39. Seleccionar la opción PROGRAMMER.

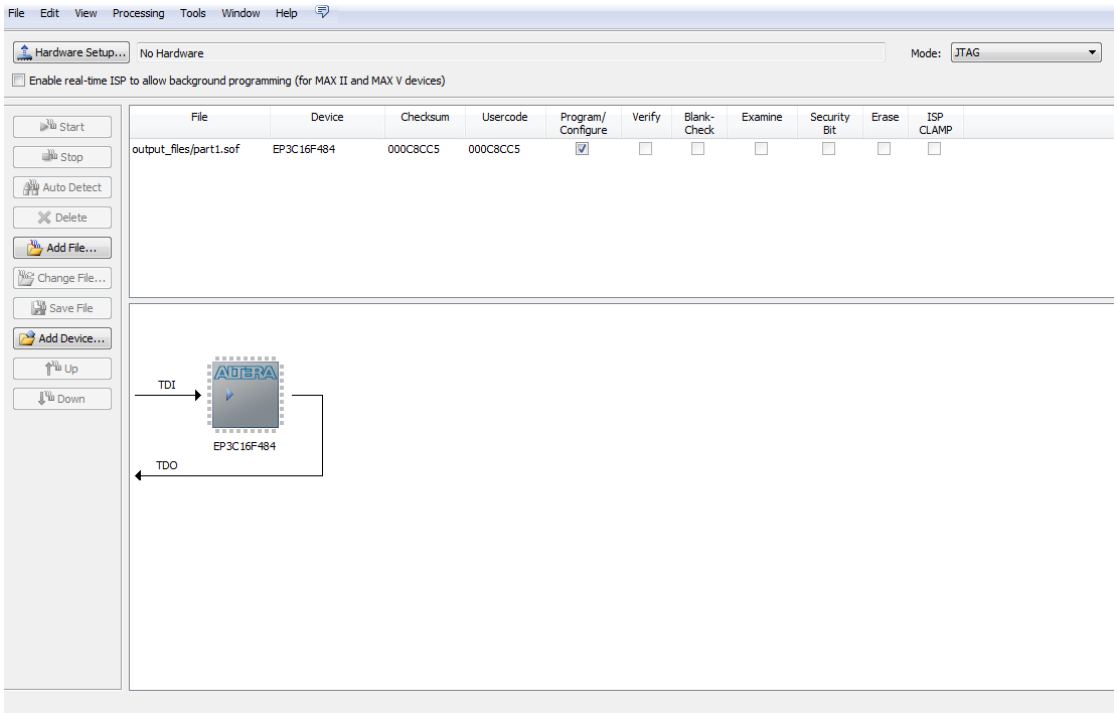


Figura 3.40. Ventana para programar la tarjeta cyclone V GX Starter.

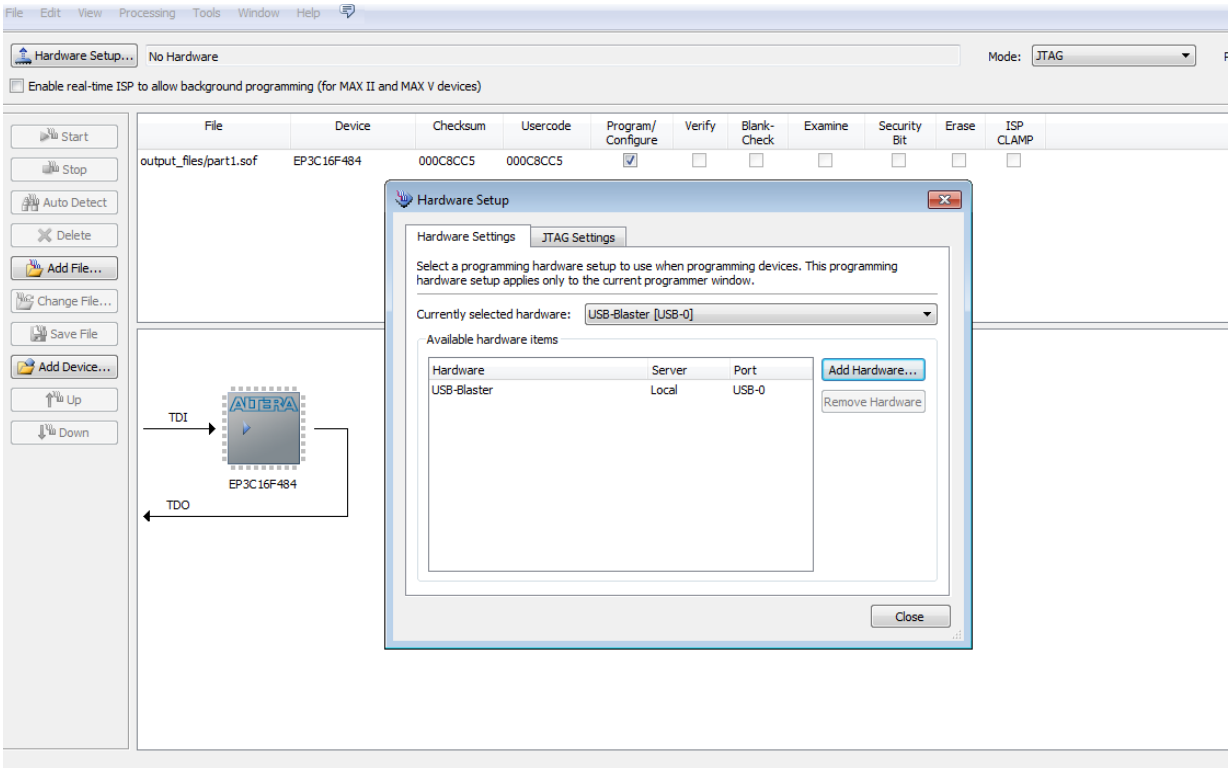


Figura 3.41. Selección del USB blaster.

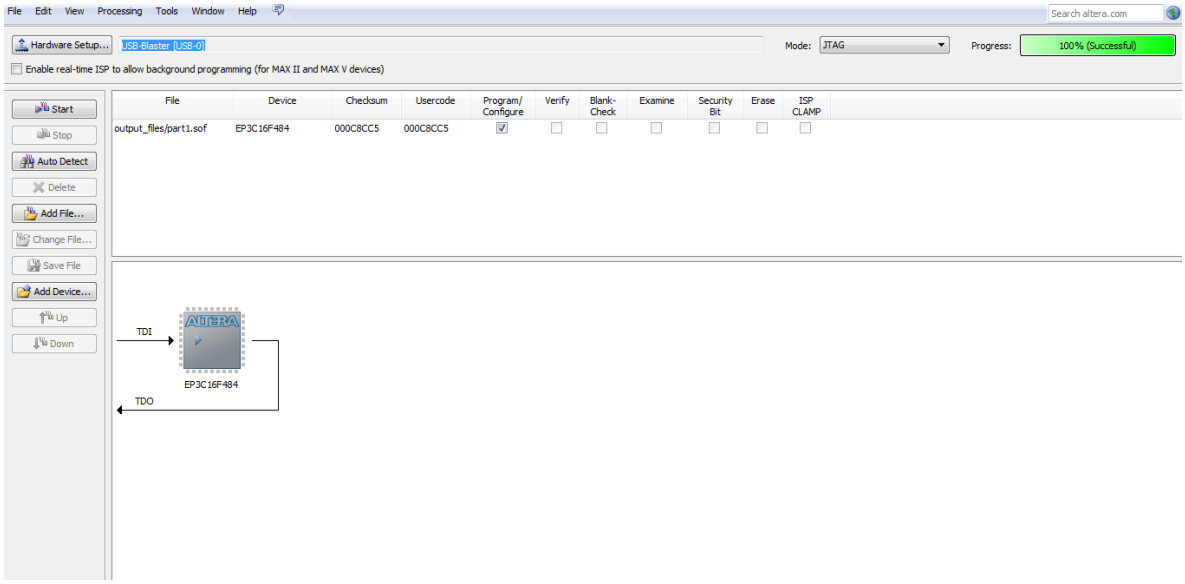


Figura 3.42. Programación Exitosa.

Capítulo IV: programación del controlador PID en VHDL.

Introducción al capítulo IV

En este capítulo se realiza la programación del controlador PID en lenguaje VHDL, el componente controlador parte de la ecuación PID para tiempo discreto y su utilización dependerá del tipo de sistema a controlar y la forma en que la señal del PID se enlace con el actuador. La señal actuante del sistema para controlar el motor de corriente continua es el PWM, por lo que la señal PID debe ejercer la acción de cambiar el ciclo de trabajo para hacer que el motor gire más rápido o se detenga. Para la programación se utilizan diferentes herramientas propias de sistemas digitales las cuales tienen fácil implementación en el chip FPGA, por ejemplo, las máquinas de estado finito, los flip flop, timmer entre otros.

Se realizan dos códigos diferentes para el control del motor (control de posición y velocidad), esto con el fin de mostrar dos aplicaciones del mismo componente controlador y reservar la mayor cantidad de switches posibles para el ingreso de las constantes proporcional, integral y derivativa con las cuales se cambiará la respuesta del sistema.

4.1 Programación del controlador PID.

En el capítulo I se desarrolló una ecuación discreta para el controlador PID, específicamente la ecuación 1.30 que se vuelve a presentar aquí como la ecuación 4.1.

$$u(t_k) = u(t_{k-1}) + q0e_k + q1e_{k-1} + q2e_{k-2} \qquad \textit{Ecuación 4.1}$$

Dónde:

$$q0 = Kp\left(1 + \frac{Td}{T}\right) \qquad \textit{Ecuación 4.1.1}$$

$$q1 = Kp\left(-1 - 2\frac{Td}{T} + \frac{T}{Ti}\right) \qquad \textit{Ecuación 4.1.2}$$

$$q2 = Kp\left(\frac{Td}{T}\right) \qquad \textit{Ecuación 4.1.3}$$

Consideraciones:

- Ya que el lenguaje VHDL es un lenguaje concurrente la mejor forma de llevar a cabo la ecuación 4.1 es mediante una máquina de estado finito (FSM) que evalúe cada termino y en el estado final obtener la respuesta de los términos en conjunto además de guardar los valores actuales para el cálculo posterior, la forma de actuar de esta máquina de estado es de forma síncrona con un reloj, de forma que conforme suceda un flanco de subida en un reloj se avance de estado. La virtud de una máquina finita para este apartado es que las

señales solo cambian en el estado que se está evaluando. Por ejemplo en el código 4.1 en la línea 94 se evalúa el estado "E1", en este estado la señal "ek" toma el valor de la señal "error" y solo cambiara su valor en el momento que el estado sea nuevamente evaluado (después de evaluar el estado "E15").

- La rampa digital es obtenida por medio de un contador de 17 bit ($2^{17} = 131072$) y un reloj de 50MHz. Se tiene una rampa con frecuencia de:

$$Frampa = \frac{50MHz}{131072} \approx 381\ Hz$$

Esta es la frecuencia de la señal PWM y su valor será comparado con la señal de salida del PID (PID_OUT) para variar su ciclo de trabajo.

- Es importante reservar estados para prevenir el efecto WINDUP y que el controlador se sature.
- La salida del controlador PID_OUT es un número que es utilizado para manejar el PWM que actuará sobre el motor por tal se limita a un número entre cero y el máximo valor que para el caso de un numero de 17 bit es de 0 a 131072, sin embargo ya que el máximo valor será comparado con una rampa de 17 bit tiene sentido limitarlo a 130000 para que existan valores en la rampa que sean mayores y establecer un tiempo mínimo de apagado de la señal PWM.

La máquina de estado finito con la que desarrollan los términos de la ecuación 4.1 se muestra en la figura 4.1.

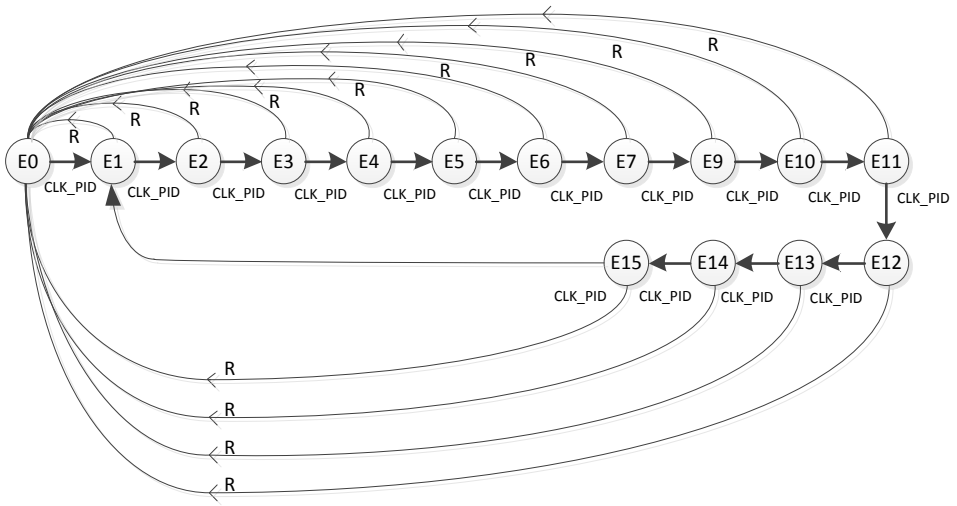


Figura 4.1 Máquina de estado finito para el componente controlador PID

Los cambios de estado de la máquina finita están en sincronía del reloj (CLK_PID). Las acciones y cálculos realizados en cada estado son:

E0= Estado de reset, líneas 85-91 del código 4.1.
E1= Guardar error actual.
E2= KP*ek.
E3=KD*T *ek.
E4=KI/T * ek-1.
E5=KP* ek-1.
E6=2*KD* ek-1.
E7=KD*T* ek-2.
E8= Acción integral <=E4.
E9= Ajuste anti Windup.
E10= Cálculo de la acción PID.
E11=Limitando la acción del controlador.
E12= Guardando la acción del controlador en la señal que se conecta a la salida PID.
E13= Guardando la acción del controlador para calculo posterior.
E14=Guardando el error antepasado ek-2.
E15=Guardando el error pasado ek-1.

El código VHDL del controlador PID (código 4.1) se presenta a continuación:

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY PID IS
7
8  PORT (
9           CLK_PID        :IN STD_LOGIC;-----------------------------------------RELOJ DE LA MAQUINA FINITA PID
10          ERROR          :IN INTEGER;-------------------------------------------ERRORPID
11          KP,KI,KD       :IN STD_LOGIC_VECTOR (6 DOWNTO 0);---------------------CONSTANTES
12          RESET          :IN STD_LOGIC;-----------------------------------------RESET ASINCRONO ACTIVO BAJO.
13          PID_OUT        :OUT INTEGER -------------------------------------------SALIDA DEL PID
14
15 );
16 END PID;
17
18
19 ARCHITECTURE behavioral OF PID IS
20
21 TYPE estado IS (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15);-----16  16 ESTADOS PARA LA FSM
22 SIGNAL actual_estado :estado:=E0;-------------------------------------------INICIAR EN E0
23 SIGNAL siguiente_estado : estado;
24
25 SIGNAL uk,u_k,uk_1,ek,ek_1,ek_2: INTEGER:=0;
26 SIGNAL m_k :integer :=0;
27 SIGNAL AUX1,AUX2,AUX3,AUX4,AUX5,AUX6 :INTEGER:=0;----------------SENALES PARA CALCULOS
28 SIGNAL A_INTEGRAL,A_I : integer:=0;-----------------------------Para Orden de la accion integral
29
30 CONSTANT T : INTEGER:= 100;
31 SIGNAL AUX_KI : INTEGER:=0;
32 SIGNAL AUX_KP:INTEGER :=0;
33 SIGNAL AUX_KD : INTEGER :=0;
34
35 BEGIN
36 AUX_KP <= TO_INTEGER(UNSIGNED (KP));
37 AUX_KD <= TO_INTEGER(UNSIGNED (KD));
38 AUX_KI <= TO_INTEGER(UNSIGNED (KI));
39
40    PROCESS (CLK_PID,RESET)
41              BEGIN
42                       IF (CLK_PID'event and CLK_PID='1') THEN
43                          IF (RESET = '0') THEN
44                             actual_estado<=E0;
45                          ELSE
46                             actual_estado<=siguiente_estado;
47                          END IF;
48                       END IF;
49              END PROCESS;
```

```
50
51
52          PROCESS (actual_estado)
53          BEGIN
54
55             CASE actual_estado IS
56                         WHEN E0 => siguiente_estado <= E1;
57                         WHEN E1 => siguiente_estado <= E2;
58                         WHEN E2 => siguiente_estado <= E3;
59                         WHEN E3 => siguiente_estado <= E4;
60                         WHEN E4 => siguiente_estado <= E5;
61                         WHEN E5 => siguiente_estado <= E6;
62                         WHEN E6 => siguiente_estado <= E7;
63                         WHEN E7 => siguiente_estado <= E8;
64                         WHEN E8 => siguiente_estado <= E9;
65                         WHEN E9 => siguiente_estado <= E10;
66                         WHEN E10 => siguiente_estado <= E11;
67                         WHEN E11 => siguiente_estado <= E12;
68                         WHEN E12 => siguiente_estado <= E13;
69                         WHEN E13 => siguiente_estado <= E14;
70                         WHEN E14 => siguiente_estado <= E15;
71                         WHEN E15 => siguiente_estado <= E1;
72                         WHEN OTHERS => siguiente_estado<=E0;
73               END CASE ;
74
75             END PROCESS;
```

```
76
77  PROCESS (CLK_PID,actual_estado,error)
78
79                 BEGIN
80
81     IF (CLK_PID'event and CLK_PID='1') THEN
82
83                 CASE (actual_estado) IS
84
85                    WHEN E0=>                                          ------------ESTADO DE RESET
86                             m_k  <= 0;
87                             u_k  <= 0;
88                             uk_1 <= 0;
89                               ek <= 0;
90                             ek_1 <= 0;
91                             ek_2 <= 0;
92                                                                  --siguiente_estado<=E1;
93
94                    WHEN E1=>
95
96                             ek <= error;
97                                                                  --siguiente_estado<=E2;
98
99                    WHEN E2=>
100
101                            AUX1<=   AUX_KP*ek;
102                                                                 --siguiente_estado<=E3;
```

```
103
104                    WHEN E3=>
105
106                             AUX2<= ((AUX_KD*T)*ek);
107                                                                    --siguiente_estado<=E4;
108
109                    WHEN E4=>
110
111                             AUX3<= ((AUX_KI*ek_1)/(T));
112                                                                    --siguiente_estado<=E5;
113
114                    WHEN E5=>
115
116                             AUX4<= AUX_KP*ek_1;
117                                                                    --siguiente_estado<=E6;
118
119                    WHEN E6=>
120
121                             AUX5<=(2*AUX_KD*T*ek_1);
122                                                                    --siguiente_estado<=E7;
123
124                    WHEN E7=>
125
126                             AUX6<=(AUX_KD*T)*ek_2;
127                                                                    --siguiente estado<=E8;
```

```
WHEN E8=>

          A_INTEGRAL <= AUX3;
                                                    --siguiente_estado<=E9;

WHEN E9=>

          IF (A_INTEGRAL > 130000 or A_iNTEGRAL <= 0) THEN--Delimitando la salida del controlador ResetWindup
               A_I<=0;
          ELSE A_I<= A_INTEGRAL;
          END IF;
                                                    --siguiente_estado<=E10;

WHEN E10=>

      u_k<=uk_1+AUX1+AUX2+A_I-AUX4-AUX5+AUX6;------u_k es un auxiliar para el calculo de la formula

                                                    --siguiente_estado<=E11;

              WHEN E11=>
                       IF (u_k) > 130000 then              --Delimitando la salida del controlador ResetWindup
                        uk<=130000;
                       elsif (u_k) <= 0 then
                        uk<=0;
                       else
                        uk<=u_k;
                       end if;
                                                --siguiente_estado<=E12;
              WHEN E12=>m_k<=uk;                --------------m_k es la salida del PID
                                               -- siguiente_estado<=E13;
              WHEN E13=>uk_1<=m_k;              --uk_1 secuencialmente toma el valor antiguo de la ecu
                                               --siguiente_estado<=E14;
              WHEN E14=>ek_2<=ek_1;             --error anterior menos 2<= error anterior menos 1
                                               --siguiente_estado<=E15;
              WHEN E15=>ek_1<=ek;               --Asignacion secuencial  error anterior menos 1 <= er
                                               --siguiente_estado<=E1;

            END CASE;
END IF;

            END PROCESS;

  PID_OUT<= M_K; ----------------------SALIDA DEL CONTROLADOR PID.

END behavioral;
```

Código 4.1. Código de la máquina de estados para el controlador PID.

Funcionamiento del componente controlador PID:

En el estado "E1" la señal "ek" toma el valor de la señal "error" la cual ha sido obtenida en el componente ERROR PID, en los estados "E2…E7" se calculan los valores $q0e_k + q1e_{k-1} + q2e_{k-2}$ mostrados en la ecuación 4.1, en el estado "E8" se guarda la acción integral (el componente de la acción del controlador que se evalúa con la constante Ki) a forma de orden para luego limitar su acción (ajuste anti Windup) en el estado "E9". En el estado "E10" se evalúa el conjunto de valores calculados en los estados anteriores para obtener el valor del controlador con la ecuación 4.1. El estado "E11" ajusta la acción total del controlador de modo que nunca sobrepase el valor establecido de 130000 para la comparación con la rampa. En los estados "E12..E15" se guardan los valores actuales para el cálculo posterior del controlador.

En el estado E12 de la máquina de estado PID se tiene una salida PID_OUT en su máximo valor (130000) producto de que existe un error máximo y por tanto la tendencia es reducirlo. Una salida PID_OUT a su máximo valor implica que la señal PWM tenga un ciclo de trabajo alto durante un periodo el cual está determinado por la frecuencia de desbordamiento del contador (cuenta_int) del módulo PWM.

Se desea aplicar el controlador antes expuesto al control de posición y velocidad de un motor dc. Para lograr este objetivo se recurre a la programación de componentes en VHDL, esto con el fin de reutilizar códigos para las dos aplicaciones

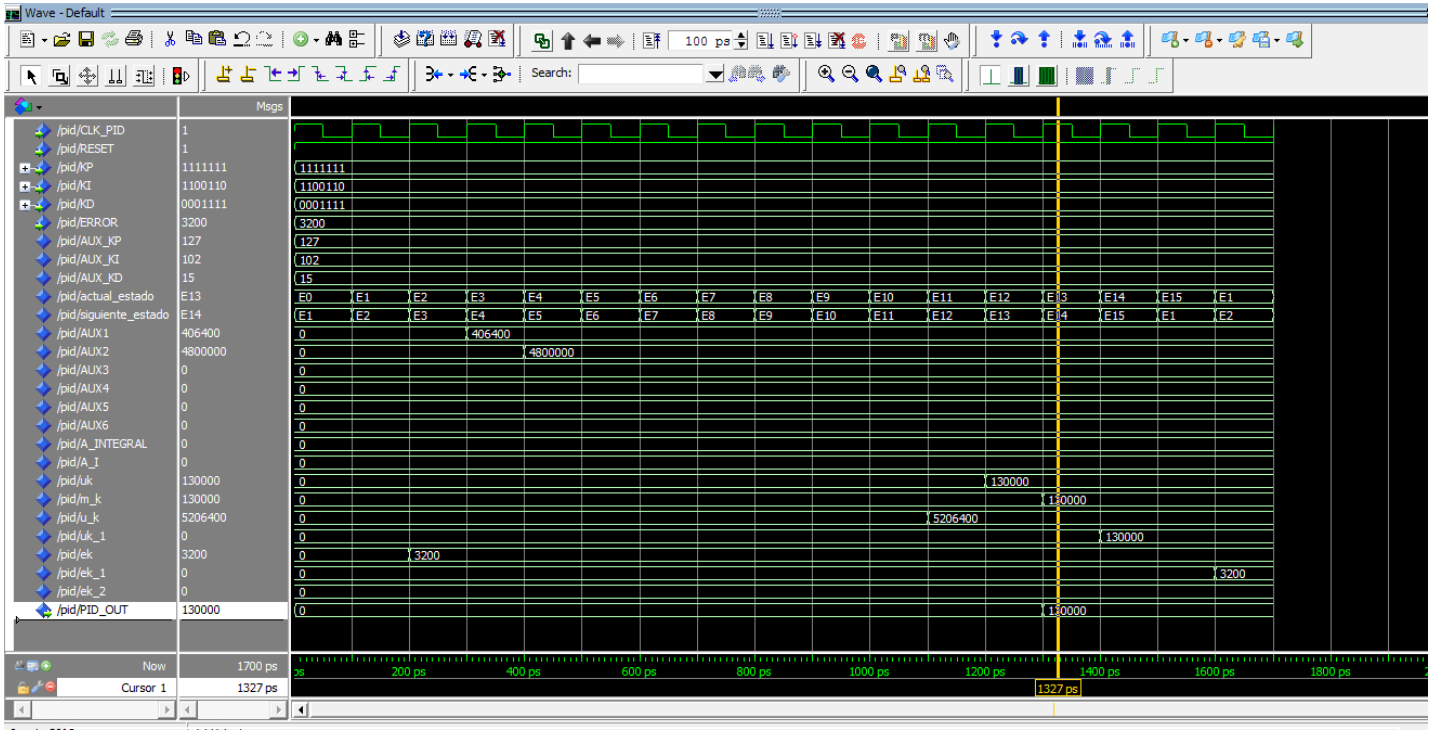
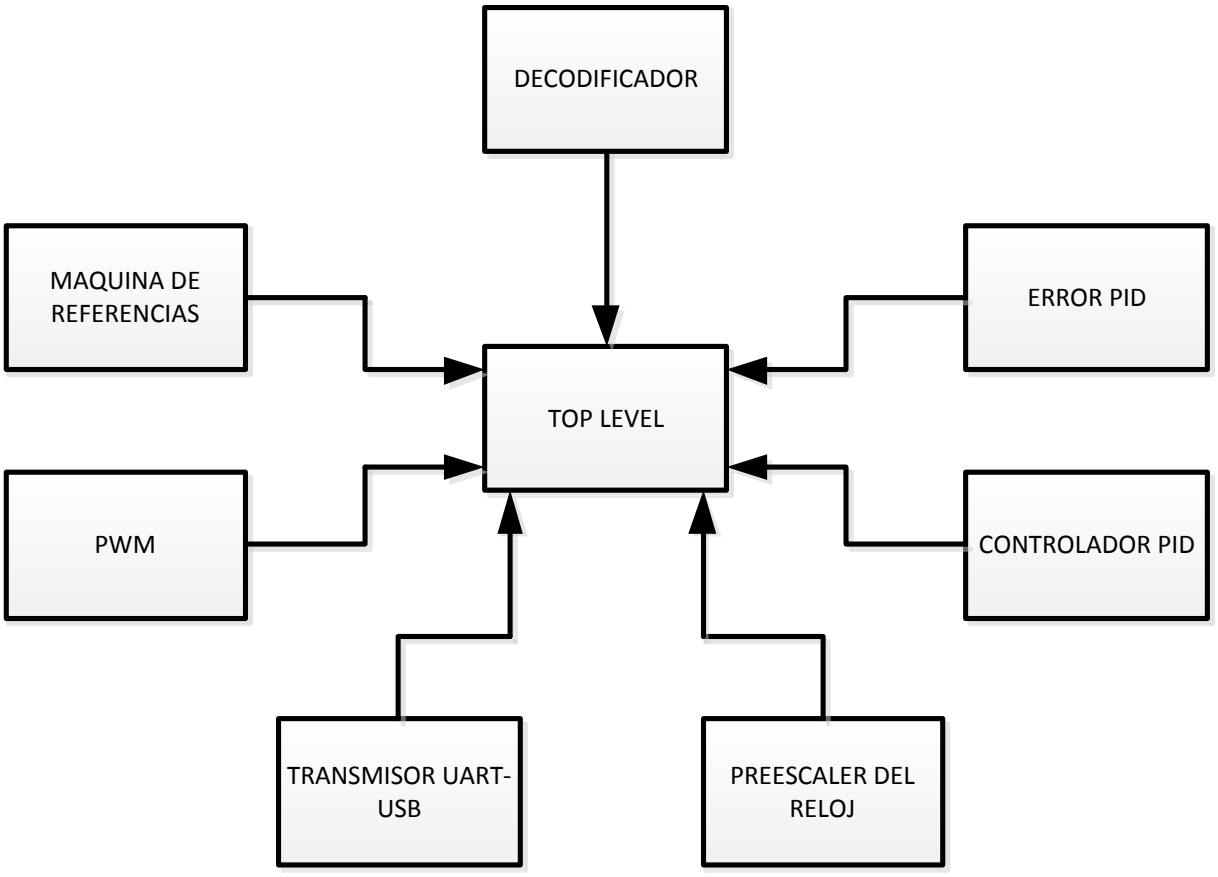


Figura 4.2. Simulación de la máquina de estados para el controlador PID

- Los componentes necesarios para el sistema son:
- Componente de referencias (FSMREF).
- Componente decodificador del encoder.
- Para posición: DECODER.
  Para velocidad: SPEED.
- Componente para cálculo del error (ERRORPID).
- Componente controlador PID (PID).
- Componente generador del PWM (PWM).
- Componente generador de frecuencias (GENFREC).
- Componente comunicación UART-USB (TRANSMISOR).

Para el uso de componentes en VHDL es necesario definir un componente jerárquico que interconecte los demás, el componente jerárquico se puede consultar en el anexo E, en el que se presentan los códigos totales de los controladores PID posición y velocidad. El esquema 4.1 muestra esta situación.



Esquema 4.1 Presentación de los componentes para el control de posición y velocidad del motor de continua. Existe un componente "TOP LEVEL" jerárquico que interconecta los módulos a fin de conseguir el sistema digital de control PID sobre el motor..

4.2 Componentes compartidos

4.2.1 Transmisor UART-USB.

Para evaluar la respuesta del sistema en el tiempo se debe programar un componente que transmita los valores obtenidos de los componentes decodificadores ante una entrada de referencia (entrada escalón), la transmisión se realizara por medio del puerto USB de la tarjeta FPGA hacia un software de la computadora que grafique los datos, ya que la tarjeta Cyclone V tiene integrado un convertidor UART-USB será necesario programar este componente transmisor para el envío de datos de forma serial asíncrona.

Se ha optado por utilizar una máquina de estado finito ver figura 4.3 para la transmisión del dato, la velocidad de transmisión es de 9600 Baudios y es recibida por el componente transmisor por la señal "BAUD" que en la figura 4.3 se muestra como el reloj clk. Adicional a la señal "BAUD" se tiene la señal "TxD_INICIO" (Tin en la figura 4.3) que maneja la máquina finita para regresar al estado inicial "0000" línea 64, la señal TxD_INICIO establece el tiempo de espaciamiento entre envío de datos a través del componente al puerto.

Si la señal "TxD_READY" está en "1" significa que el transmisor está listo para recibir un nuevo dato (está en el estado 0000) esto también quiere decir que la señal "TxD_BUSSY" está en "0". A lo largo de los demás estados el transmisor inhabilita la recepción de nuevos datos por medio de la señal "TxD_BUSSY=1" Y la máquina finita se encarga de enviar bit a bit los 8 bit por medio de la señal "TxD" al convertidor además de los bit de inicio y de paro (líneas 99-111).

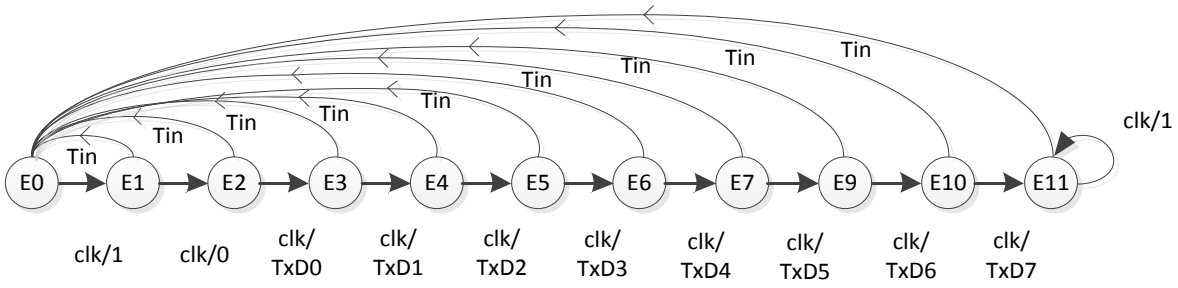


Figura 4.3 Máquina de estado finito para la transmisión del dato de forma serial asíncrona. Clk establece el tiempo de envío de bit y Tin establece el tiempo de envío de byte cuando espera en el estado E11 por un nuevo dato.

El código correspondiente al transmisor es el código 4.2. Una simulación en el programa Modelsim Muestra la transmisión de 8 bit "10110100" por medio de la señal "TxD" a una velocidad determinada por la señal "BAUD" después del bit de inicio "0" y finalizando con el bit de paro "1" ver figura 4.4.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY TRANSMISOR IS
7
8  PORT (
9             BAUD        : IN STD_LOGIC;
10            TxD_INICIO  : IN STD_LOGIC;
11            TxD_DATO    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
12            TxD         : OUT STD_LOGIC;
13            TxD_BUSY    : OUT STD_LOGIC
14 );
15
16 END TRANSMISOR;
17
18
19 ARCHITECTURE BEHAVIORAL OF TRANSMISOR IS
20
21 SIGNAL TxD_AUX_DATO : STD_LOGIC_VECTOR (7 DOWNTO 0);
22 SIGNAL STATE : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
23 SIGNAL NEXT_STATE : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
24 SIGNAL TxD_READY : STD_LOGIC;
25
26 BEGIN
27
28 PROCESS (STATE)
29 BEGIN
30 CASE STATE IS
31
32          WHEN "0000" => TxD_READY <= '1';
33          WHEN OTHERS => TxD_READY <= '0';
34
35 END CASE;
36
37 END PROCESS;
38
39 PROCESS (STATE)
40 BEGIN
41 CASE STATE IS
42
43          WHEN "0000" => TxD_BUSY <= '0';
44          WHEN OTHERS => TxD_BUSY <= '1';
45
46 END CASE;
47
48 END PROCESS;
50 PROCESS (TxD_READY)
51 BEGIN
52 IF TxD_READY'EVENT AND TxD_READY ='1' THEN
53
54 TxD_AUX_DATO <= TxD_DATO;
55
56 END IF;
57
58 END PROCESS;
59
60 PROCESS (BAUD)
61 BEGIN
62 IF BAUD'EVENT AND BAUD = '1' THEN
63
64    IF TxD_INICIO = '0' THEN
65          STATE <= "0000";
66    ELSE
67          STATE<= NEXT_STATE;
68    END IF;
69 END IF;
70
71 END PROCESS;
```

```
73  PROCESS (STATE)
74  BEGIN
75  CASE STATE IS
76
77     WHEN "0000" => NEXT_STATE <= "0001";
78     WHEN "0001" => NEXT_STATE <= "0010";
79     WHEN "0010" => NEXT_STATE <= "0011";
80     WHEN "0011" => NEXT_STATE <= "0100";
81     WHEN "0100" => NEXT_STATE <= "0101";
82     WHEN "0101" => NEXT_STATE <= "0110";
83     WHEN "0110" => NEXT_STATE <= "0111";
84     WHEN "0111" => NEXT_STATE <= "1000";
85     WHEN "1000" => NEXT_STATE <= "1001";
86     WHEN "1001" => NEXT_STATE <= "1010";
87     WHEN "1010" => NEXT_STATE <= "1011";
88     WHEN "1011" => NEXT_STATE <= "1011";
89     WHEN OTHERS => NEXT_STATE <= "1011";
90
91  END CASE;
92
93  END PROCESS;
94
95  PROCESS (STATE,TXD_AUX_DATO)
96  BEGIN
97  CASE STATE IS
98
99     WHEN "0000" => TxD  <= '1' ;
100    WHEN "0001" => TxD  <= '0' ;
101    WHEN "0010" => TxD  <= TxD_AUX_DATO(0);
102    WHEN "0011" => TxD  <= TxD_AUX_DATO(1);
103    WHEN "0100" => TxD  <= TxD_AUX_DATO(2);
104    WHEN "0101" => TxD  <= TxD_AUX_DATO(3);
105    WHEN "0110" => TxD  <= TxD_AUX_DATO(4);
106    WHEN "0111" => TxD  <= TxD_AUX_DATO(5);
107    WHEN "1000" => TxD  <= TxD_AUX_DATO(6);
108    WHEN "1001" => TxD  <= TxD_AUX_DATO(7);
109    WHEN "1010" => TxD  <= '1';
110    WHEN "1011" => TxD  <= '1';
111    WHEN OTHERS => TxD  <= '1';
112
113 END CASE;
114
115 END PROCESS;
116
117 END BEHAVIORAL;
```

Código 4.2. Código del transmisor serial Asíncrono.

4.2.2 Generación del PWM

La salida del componente PID "PID_OUT" es un numero binario de 17 bit que debe manejar la señal PWM, la forma de enlazar estas 2 señales es por medio de una comparación entre los valores de la salida del componente PID (PID_OUT) y el contador (cuenta_int) del módulo PWM, de modo que siempre que el valor del PID sea mayor que el contador la señal PWM se mantendrá en "1". La señal PID_OUT se ha limitado a 130000 (el máximo valor es 131072) de forma que en un periodo de desborde del contador (la cuenta máxima es 131072 luego de esto se vuelve 0, cuenta_int) siempre exista un tiempo en el que el contador (cuenta_int ver línea 32) sea mayor que PID_OUT de forma que haya un tiempo de apagado de la señal PWM, para mayor referencia ver sección 1.7.1.

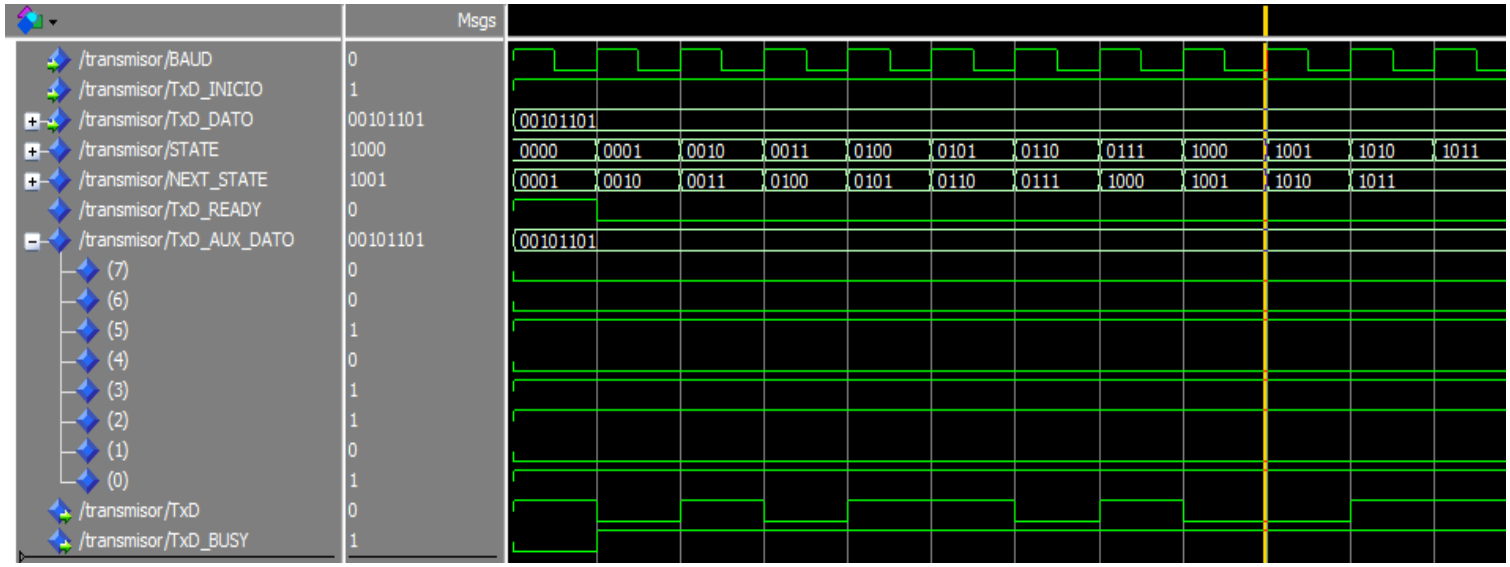


Figura 4.4 Simulación del transmisor serial asíncrono, se transmite el Byte "00101101", el byte se transmite desde el LSB hacia el MSB respectivamente.

El código del componente PWM (código 4.3) se muestra a continuación.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.NUMERIC_STD.ALL;
4  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  ENTITY PWM IS
7
8  PORT (
9               clk_50     : IN  STD_LOGIC;
10              PWM_IN     : IN INTEGER;-----------------es la salida del PID
11              SALIDA     : OUT STD_LOGIC---------------- es la senial actuante PWM
12        );
13
14 END PWM;
15
16 ARCHITECTURE Behavioral OF PWM IS
17
18 SIGNAL cuenta_int    : std_logic_vector(16 downto 0) := (others => '0');
19 SIGNAL  porcentaje : std_logic_vector(16 downto 0):=(OTHERS=>'0');
20
21 BEGIN
22
23 porcentaje <= std_logic_vector(to_unsigned(PWM_IN, 17));
25 P1: PROCESS (clk_50)
26  BEGIN
27
28        IF (clk_50'EVENT and clk_50 = '1' ) THEN
29           IF cuenta_int = "11111111111111111" THEN-----------------------99%
30              cuenta_int<=( OTHERS => '0');
31           ELSE
32              cuenta_int <= cuenta_int+1;-------------------------
33           END IF;
34        END IF;
35
36 END PROCESS;
37
38 salida <= '0' WHEN (cuenta_int < porcentaje) ELSE '1';---------------se rige por porcentajes.
39
40 END Behavioral;
```

Código 4.3. Código del componente PWM. Generador de rampa

Funcionamiento del componente PWM:

Por cada flanco de subida de un pulso del reloj de 50Mhz el contador se incrementa en uno esto sucede en la línea 32 del código 4.3, el número máximo que puede contarse esta limitado por el número de bit del contador, de forma que cuando los 17 bit de la señal “cuenta_int” sean igual a “1” el contador se reiniciará a cero. En la línea 38 del código 4.3 la señal salida (que es en realidad la señal PWM de salida que controla el motor) cambia su ciclo de trabajo manteniéndose en “1” siempre y cuando cuenta_int < porcentaje.

Simulación del componente PWM:

En la simulación (ver figura 4.5) se muestra las diferencias entre el ciclo activo de la señal PWM (SALIDA) para dos valores diferentes de PWM_IN. Cabe mencionar que ya que se aísla la tarjeta Cyclone V GX por medio de un optoacoplador de colector abierto, la señal hacia el integrado L298N se mantendrá en alto mientras la señal “SALIDA=0”

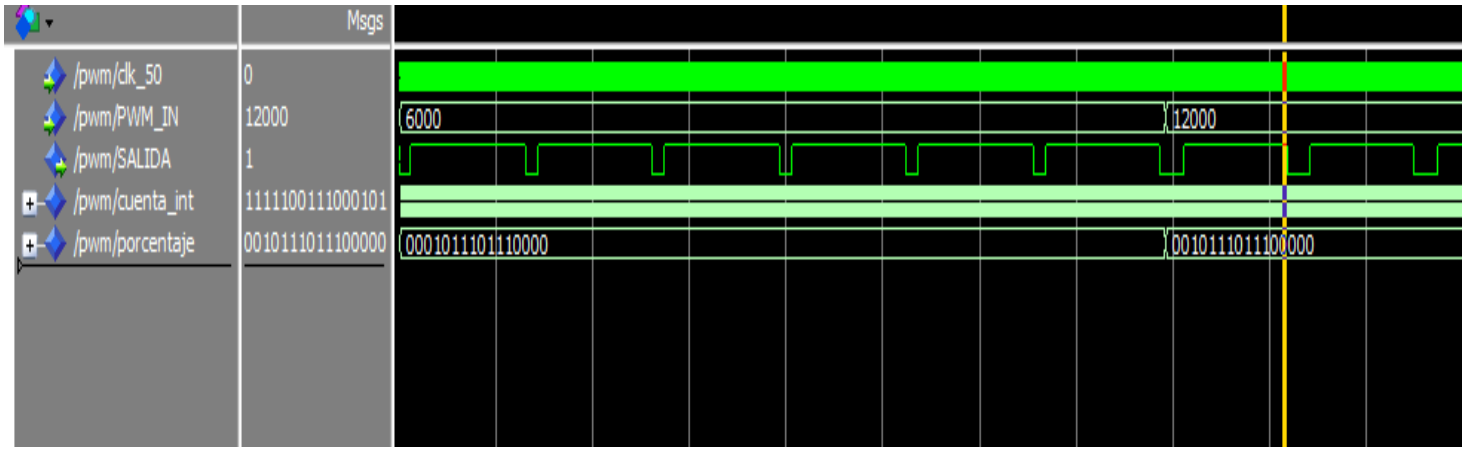


Figura 4.5 Simulación del componente PWM, en la figura se muestra una variación en el ciclo de trabajo de la señal "SALIDA"

4.2.3 Generador de frecuencias (GENFREC).

Algunos componentes del código utilizan señales de reloj que no se obtienen directamente de la tarjeta, por ejemplo el reloj de la maquina finita PID, el componente de trasmisión de datos serial. Ya que estas señales (señales de reloj) son necesarias para el funcionamiento adecuado de los componentes se optó por crear un componente que genere señales de reloj a diferentes frecuencias. La lógica de programación es sencilla. Si se tiene un reloj de 50MHz entonces el número de conteos necesarios para generar una frecuencia particular es:

$$Conteos = 50MHz/FREC$$

A modo de ejemplo si se desea una señal de 2kHz entonces se deben realizar 25000 conteos con el reloj de 50Mhz.

$$Conteos = \frac{50MHz}{2000Hz} = 25000\ conteos$$

Por lo tanto si se desea una señal cuadrada simétrica, 12500 conteos permanecerá el nivel lógico bajo “0” y 12500 conteos en nivel lógico alto “1”

El código VHDL que realiza esta labor es mostrado a continuación:

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.NUMERIC_STD.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY GEN_FREC IS
7   PORT
8   (
9         CLK_50      : IN STD_LOGIC;
10        F_BASE      : IN INTEGER;
11        FREC_OUT    : OUT STD_LOGIC
12
13  );
14  END GEN_FREC;
15
16
17  ARCHITECTURE BEHAVIORAL OF GEN_FREC IS
18
19  SIGNAL COUNT1   : INTEGER := 0;
20  SIGNAL CLK2     : STD_LOGIC:='0';
21
22  BEGIN
```

```
24  PROCESS (CLK_50)
25  BEGIN
26
27        IF CLK_50'EVENT AND CLK_50 ='1' THEN
28
29              IF COUNT1 >= F_BASE THEN
30                 COUNT1<=0;
31                 CLK2<=NOT CLK2;
32              ELSE
33                 COUNT1<=COUNT1+1;
34              END IF;
35        END IF;
36  END PROCESS;
37
38  FREC_OUT<=CLK2;
39
40  END BEHAVIORAL;
41
```

Código 4.4. Código pre escalador del reloj, este genera señales cuadradas a diferentes frecuencias.

La señal "F_BASE" es recibida como la mitad del periodo de la señal cuadrada de modo que mientras que la señal "COUNT1" no supere el valor de "F_BASE" el valor del reloj "CLK2" será cero y cuando lo supere el valor de "CLK2" será uno. Dando lugar a una señal cuadrada simétrica, una simulación del componente pre escalador del reloj se muestra en la figura 4.6.

4.3 Control de posición.

4.3.1 Máquina finita de referencias (FSM_REF).

Ya que se utilizará un método heurístico (ver sección 1.5) para la sintonización del PID es necesario reservar la mayor cantidad de switches de la tarjeta Cyclone V para tener un amplio rango de valores para las constantes KP, KD, KI. Además de esto el ingreso de las referencias será mediante un código binario con el fin de aumentar la facilidad del ingreso.

Para realizar las tareas anteriores se programa una máquina de estado finito (ver figura 4.7) en la que su reloj es controlado por medio de un pulsador de la tarjeta.

Una descripción las acciones de los estados de la máquina de referencias para posición:

E0= estado de reset, se borra la referencia y el valor de las constantes.

E1= se lee el estado lógico de los switches (switches 0…1) de la tarjeta a fin de obtener un código binario que por medio de código se hace corresponder a una referencia, ver líneas 84-88 del código 4.5.
E2= se lee el estado lógico de los switches (switches 7…2) de manera que este número binario es convertido a decimal y representara el valor de la constante KP ver líneas 95-97 del código 45.

E3= se lee el estado lógico de los switches (switches 7…2) de manera que este número binario es convertido a decimal y representara el valor de la constante KI.

.

Simulación del componente GENFREC

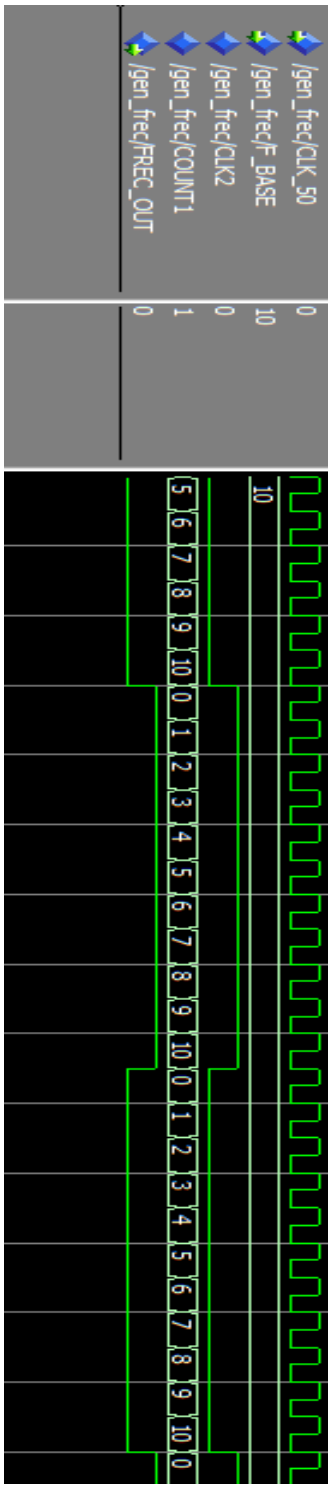


Figura 4.6. Simulador del componente GENFREC, generador de señales cuadradas.

E4= se lee el estado lógico de los switches (switches 7…2) de manera que este número binario es convertido a decimal y representara el valor de la constante kd.

E5= con el fin de que los datos sean transmitidos a los demás componentes al mismo tiempo, en el estado E5 toman valor las salidas del componente FSMREF.

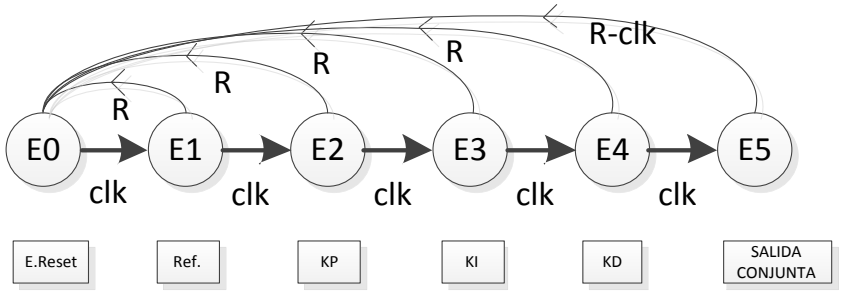


Figura 4.7 máquina de estado finito para las referencias. A medida se avanza en los estados de la maquina se van guardando los valores de referencia y contantes kp, ki, kd.

El código de la máquina finita de referencias para control de posición (código 4.5) se presenta a continuación:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY FSMREF IS

PORT (

            RESET       :  IN STD_LOGIC;-----------------------RESET DE SELECCION TIENE QUE SER UN SW.
            SEL         :  IN STD_LOGIC_VECTOR (1 DOWNTO 0);---SELECCION DE LA REFERENCIA 00,01,10...
            CLK_fSM_REF :  IN STD_LOGIC;-----------------------RELOJ DE LA MAQUINA DE REFERENCIA
            CONSTANTES  :  IN STD_LOGIC_VECTOR (5 DOWNTO 0);
            REFERENCIA  :  OUT    INTEGER;---------------------REFERENCIA DE SALIDA PARA CALCULO DEL ERROR
            ENPOS       :  OUT    STD_LOGIC;-------------------HABILITADOR, POSICION,VELOCIDAD.
            KP,KI,KD    :  OUT    STD_LOGIC_VECTOR (5 DOWNTO 0);---- CONSTANTES DE SALIDA
            LED_INDICADOR: OUT    STD_LOGIC_VECTOR (5 DOWNTO 0)----- INDICAN LOS ESTADOS DE LA MAQUINA

);

END FSMREF;
```

```
ARCHITECTURE behavioral OF FSMREF IS

TYPE estado IS (E0,E1,E2,E3,E4,E5);-----6 estados para la FSM
SIGNAL actual_estado :estado:=E0;-----------------------------------------------Iniciar en el estado 0
SIGNAL siguiente_estado : estado;
SIGNAL AUX_KP,k_p,AUX_KI,k_i,AUX_KD,k_d : std_LOGIC_VECTOR(5 DOWNTO 0):="000000";
SIGNAL AUX_REF_OUT, REF_OUT : INTEGER:=0;
SIGNAL AUX_EN0 : STD_LOGIC:= '0';
SIGNAL LED :STD_LOGIC_VECTOR (5 DOWNTO 0):="000000";

BEGIN

   PROCESS (CLK_fSM_REF,RESET)
               BEGIN
                        IF (CLK_fSM_REF'event and CLK_fSM_REF='1') THEN
                           IF (RESET = '0') THEN
                              actual_estado<=E0;
                           ELSE
                              actual_estado<=siguiente_estado;
                           END IF;
                        END IF;
   END PROCESS;
```

```
47    PROCESS (actual_estado)
48                BEGIN
49                   CASE actual_estado IS
50                         WHEN E0 => siguiente_estado <= E1;
51                         WHEN E1 => siguiente_estado <= E2;
52                         WHEN E2 => siguiente_estado <= E3;
53                         WHEN E3 => siguiente_estado <= E4;
54                         WHEN E4 => siguiente_estado <= E5;
55                         WHEN E5 => siguiente_estado <= E0;
56                         WHEN OTHERS => siguiente_estado<=E0;
57                    END CASE ;
58    END PROCESS;
59
60
61  PROCESS (CLK_fSM_REF,ACTUAL_ESTADO)
62
63  BEGIN
64
65        IF (CLK_fSM_REF'event and CLK_fSM_REF='1' ) THEN
66
67           CASE (actual_estado) IS
68
69                    WHEN E0=>               ------------ESTADO DE RESET
70
71                             k_p<="000000";
72                             k_d<="000000";
73                             k_i<="000000";
74                             AUX_EN0<='0';
75                             REF_OUT<=0;
76                             LED<="000001";
77
78                    WHEN E1=>
79
80                    LED(0)<='0';
81
82                         CASE SEL IS
83
84                                                 WHEN "00" =>AUX_REF_OUT <=  800;-----45 GRADOS
85                                                 WHEN "01" =>AUX_REF_OUT <= 1600;-----90 GRADOS
86                                                 WHEN "10" =>AUX_REF_OUT <= 3200;-----180 GRADOS
87                                                 WHEN "11" =>AUX_REF_OUT <= 4444;-----250 GRADOS
88                                                 WHEN OTHERS=>AUX_REF_OUT <= 0;-------0 GRADOS
89
90                         END CASE;
91
92                    LED(1)<='1';
93
94                    WHEN E2=>
95
96                             AUX_KP<=CONSTANTES;
97                             LED(2) <='1';
98
99                    WHEN E3=>
100
101                            AUX_KD<=CONSTANTES;
102                            LED(3) <='1';
103                   WHEN E4=>
104
105                            AUX_KI<=CONSTANTES;
106                            LED(4) <='1';
107
108                   WHEN E5=>
109
110                      k_p<=AUX_KP;
111                      k_d<=AUX_KD;
112                      k_i<=AUX_KI;
113                      REF_OUT<=AUX_REF_OUT;
114                      AUX_EN0<='1';
115                      LED(5) <='1';
116
117              END CASE;
118         END IF;
119
120                 END PROCESS;
121
122   KP<=K_P;
123   KI<=K_I;
124   KD<=K_D;
125   ENPOS<=AUX_EN0;
126   REFERENCIA <= REF_OUT;
127   LED_INDICADOR<=LED;
128
129   END behavioral;
```

Código 4.5. Código de la máquina finita para obtención de referencias y constantes del PID en el caso del control de posición.

Funcionamiento del componente FSMREF (maquina finita de referencias para el control de posición)

La máquina de estado finito avanza desde el estado inicial E0 hasta el estado final E5 cuando sucede un flanco positivo del reloj CLK_FSM_REF (línea 37) que es conectado al KEY0 de la tarjeta Cyclone V, a medida que la máquina avanza de estado va guardando el numero binario que encuentra en los Switch asignados en la señal particular que se evalúa en ese momento.

Se ha configurado la señal "SEL" de modo que esta es ingresada por medio de los Switch SW1..SW0, este número puede tener 4 valores posibles y dependiendo de su valor se escoge una referencia para la posición angular, esto se aprecia en el estado "E2" y las líneas 84-88.

Posterior al ingreso de la referencia se procede con el ingreso de las constantes, la señal de entrada "CONSTANTES" se ha configurado de modo que toma su valor binario de los Switch SW7…SW2 y conforme la máquina cambia de estado asigna este valor binario a las señales "AUX_KP, AUX_KD, AUX_KI" de esta forma se utilizan los mismos Switch para asignar diferentes valores a las constantes del PID.

En el estado "E5" se guarda la referencia y los valores de las constantes ingresados en los datos anteriores todos en un mismo instante de tiempo, esto para que cuando se inicien los otros componentes cuenten con todos los datos de forma inmediata. Para facilitar el ingreso de los datos y determinar el estado de la máquina finita se utiliza una señal "LED_INDICADOR" que está conectada a los LEDG5…LEDG0 de la tarjeta y cambian conforme se está en el estado E5..E0 respectivamente.

Simulación del componente FSMREF para el control de posición:

El funcionamiento de la máquina finita de referencia es verificado mediante una simulación en el programa Modelsim de Altera. Se muestra en la figura 4.8

4.3.2 Decodificador de posición (DECODER).

En el capítulo I se presentó el decodificador en modo 4x (sección 1.10.2) para un encoder incremental en cuadratura. La finalidad de este era aumentar la resolución del encoder que viene acoplado al motor de modo que de 1600PPR pase a ser 6400PPR.

Las referencias fueron calculadas también en el capítulo I (sección 1.10.3) y son citadas a continuación:

$$Ref45^o = 17.77 * 45 \quad = 800 \ \ conteos$$
$$Ref90^o = 17.77 * 90 \quad = 1600 \ conteos$$
$$Ref180^o = 17.77 * 180 = 3200 \ conteos$$
$$Ref220^o = 17.77 * 220 = 3909 \ conteos$$

A modo de ejemplo esto quiere decir que si se tiene una referencia de 180 grados, entonces el contador del decodificador debe contar 3200 pulsos para reducir el error a cero.

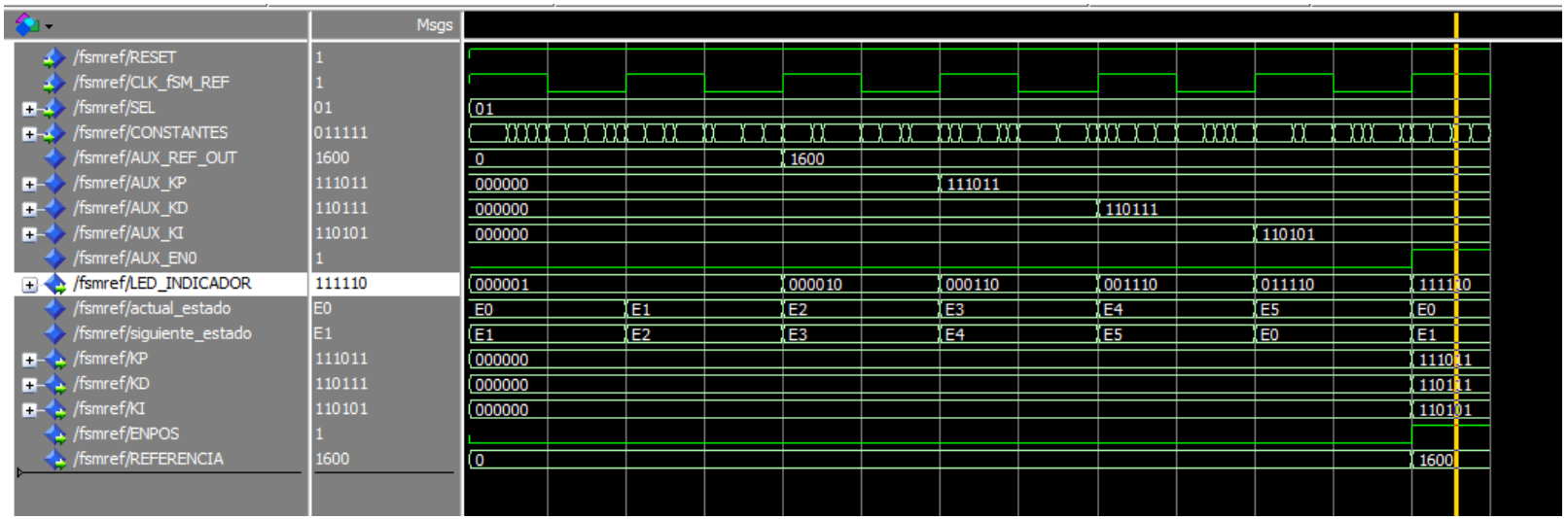


Figura 4.8. Simulación de la máquina finita para la obtención de constantes y referencias.

El código del decodificador de posición DECODERPOS (código 4.6) se presenta a continuación:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;


ENTITY DECODER IS
    PORT ( CLK_50, quadA, quadB, RESET : IN STD_LOGIC;
           GRADOS : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
           FB:      INOUT INTEGER----------------------REALIMENTANCION DEL ENCODER FEEDBACK
         );
END decoder;

ARCHITECTURE Behavioral OF decoder IS

SIGNAL  Q1, Q2, Q3, Q4, Q5, Q6 : STD_LOGIC := '0' ;
SIGNAL COUNT,AUX_LED  : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
SIGNAL AUX_FB, AUX_GRADOS : INTEGER := 0;
SIGNAL DIR, CE : STD_LOGIC:='0';-------------------------CE = conut enable, DIR = direccion

BEGIN

PROCESS (CLK_50, quadA, quadB) BEGIN
   IF (CLK_50'event and CLK_50='1') THEN
         Q1<=quadA;
   END IF;
END PROCESS;


PROCESS (CLK_50, quadA, quadB) BEGIN
   IF (CLK_50'event AND CLK_50='1') THEN
         Q2<=quadB;
   END IF;
END PROCESS;

PROCESS (CLK_50,Q1) BEGIN
   IF (CLK_50'event AND CLK_50='1') THEN
      Q3<=Q1;
   END IF;
END PROCESS;


PROCESS (CLK_50,Q2) BEGIN
   IF (CLK_50'event AND CLK_50='1') THEN
         Q4<=Q2;
      END IF;
END PROCESS;

PROCESS (CLK_50,Q3) BEGIN
   IF (CLK_50'event AND CLK_50='1') THEN
         Q5<=Q3;
   END IF;
END PROCESS;


PROCESS (CLK_50,Q4) BEGIN
   IF (CLK_50'event AND CLK_50='1') THEN
         Q6<=Q4;
   END IF;
END PROCESS;


DIR <= Q3 XOR Q6;
CE  <=  Q3 XOR Q4 XOR Q5 XOR Q6;
```

```
65
66  PROCESS(CE,CLK_50) BEGIN
67
68        IF CLK_50'EVENT AND CLK_50 = '1' THEN
69                  IF RESET = '0' THEN
70                     COUNT<=(OTHERS => '0');
71                  ELSE
72                     IF (CE='1') THEN
73                        IF DIR = '1' THEN
74                           COUNT<=COUNT+1;
75                        ELSE
76                           COUNT<=COUNT-1;
77                        END IF;
78                     END IF;
79                  END IF;
80        END IF;
81
82  END PROCESS;
83
84  AUX_FB <= TO_INTEGER(UNSIGNED(COUNT));----------------------CONTADOR EN MODO 4X
85  FB<=AUX_FB;                           ----------------------FEEDBACK EN MODO INTEGER
86
87  AUX_GRADOS <= (AUX_FB*360)/6400;----------------------------CALCULO DE GRADOS PARA EL TRANSMISOR
88  GRADOS <= std_logic_vector(to_unsigned(AUX_GRADOS, 8));----CONVERSION A BINARIO DE 8 BIT PARA EL TRANSMISOR
89
90
91  END Behavioral;
```

Código 4.6. Código decodificador de posición a partir de las señales en cuadratura.

La simulación del componente decodificador de posición DECODERPOS en Modelsim de Altera se presenta en la figura 4.9.

De la simulación se observa que siempre que la señal “CE” coincida con el estado alto de “DIR” entonces el decodificador contará en modo ascendente, esto siempre sucederá mientras el quadA del encoder adelante al quadB en 90 grados. Esto lleva a que por facilidad los grados que gira el motor se midan en sentido horario por lo tanto negativos.
En la figura 4.9 se muestra el incremento gradual desde cero hasta 41 en la señal de salida “FB” por lo que para alcanzar una referencia de 180 grados (3200 conteos) falta contar 3159 y así reducir el error a cero.

4.3.3 Cálculo del error (ERRORPID) para el control de posición.

Debido a que el error se va minimizando a lo largo del tiempo conforme el motor gira hasta la referencia de posición angular seleccionada, puede llegar a un punto donde el error es negativo, esto es que la resta de la referencia menos el Feedback sea negativa.

Cuando ocurre el error negativo para el caso del control de posición, la señal PWM reducirá su valor para reducirlo a cero, sin embargo esto no hará ningún efecto al error pues se seguirá incrementando a medida que el eje del motor gira.

Para corregir el sobrepaso se recurre a ordenar mediante las señales “DR, IZ” el giro del motor en sentido contrario, esto está contemplado en el decodificador ya que si el motor gira en sentido contrario CCW entonces se produce un decremento en el Feedback y así se reducirá el error desde negativo a cero, de hecho las oscilaciones posibles en la sintonización son muestra clara de este conjunto de correcciones.

La figura 4.10 muestra el esquema de control de posición del motor de continua.

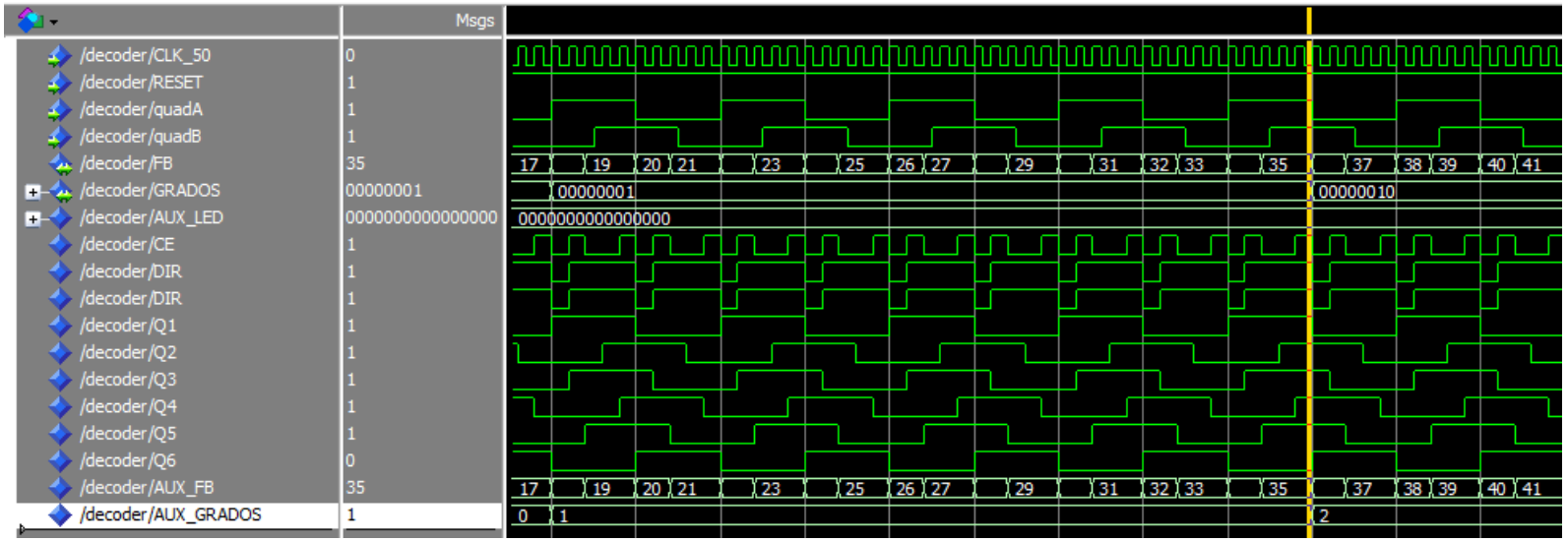


Figura 4.9. Simulación del decodificador de posición.

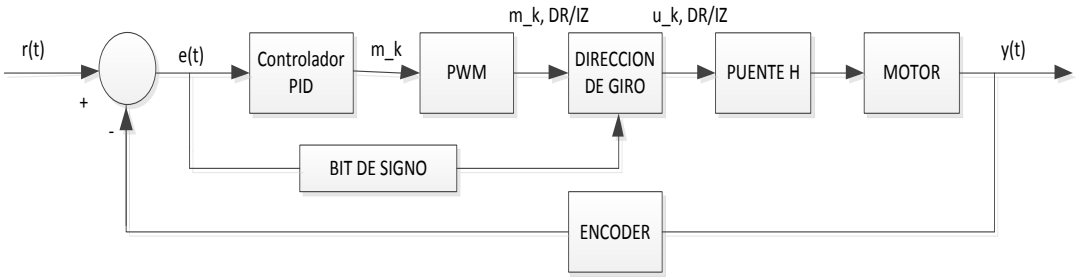


Figura 4.10 diagrama de bloques del control PID de posición para el motor de continua.

Ya que se utilizaran las señales de mando "DR, IZ" para cambiar el giro del motor se quiere que el error sea rápidamente reducido a cero por lo que se opta por enviar el valor absoluto del error al componente PWM, esto se realiza en las líneas 67 a la 78.

El componente (ERROPID) para el control de posición se presenta en el código 4.7.

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   USE IEEE.NUMERIC_STD.ALL;
4   USE IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   ENTITY ERRORPID IS
7
8   PORT(
9            CLK_50          : IN STD_LOGIC;
10           REFERENCIA      : IN INTEGER;
11           RESET           : IN STD_LOGIC;
12           FB              : IN INTEGER;
13           ENPOS           : IN STD_LOGIC;
14           ERROR           : OUT INTEGER;
15           IZ,DR           : OUT STD_LOGIC
16  );
17  END ERRORPID;
18
19  ARCHITECTURE behavioral OF ERRORPID IS
20
21  SIGNAL AUX_ERROR : INTEGER :=0;
22  SIGNAL AUX_IZ,AUX_DR :STD_LOGIC :='1';
23
24  BEGIN
25
26  PROCESS (FB,clk_50) BEGIN
27
28        IF CLK_50'EVENT AND CLK_50 ='1' THEN
29
30              IF RESET = '0' THEN
31
32                    AUX_ERROR<=0;
33                    AUX_DR<='1';
34                    AUX_IZ<='1';
35
36              ELSE
37
38                    IF ENPOS='1' THEN
```

```
39
40                              AUX_ERROR<=REFERENCIA - FB;
41
42                      IF AUX_ERROR < 0 THEN
43
44                              AUX_IZ<='0';--------------giro a la izquierda
45                              AUX_DR<='1';--------------CCW
46
47                      ELSIF (AUX_ERROR > 15 AND AUX_ERROR <6400 ) THEN
48
49                              AUX_DR<='0';--------------giro a la derecha CW
50                              AUX_IZ<='1';
51                      ELSE
52                              AUX_DR<='1';--------------DETENER MOTOR
53                              AUX_IZ<='1';
54                      END IF;
55
56                      END IF;
57
58
59                  END IF;
60
61      END IF;
62
63  END PROCESS;
64
65  PROCESS (CLK_50) BEGIN
66
67      IF CLK_50'EVENT AND CLK_50='1' THEN
68
69          IF AUX_ERROR < 0 THEN
70
71              ERROR<= AUX_ERROR*(-1);
72
73          ELSE
74
75              ERROR<=AUX_ERROR;
76
77          END IF;
78      END IF;
79
80  END PROCESS;
81
82  DR<=AUX_DR;
83  IZ<=AUX_IZ;
84
85
86  END behavioral;
```

Código 4.7. Código encargado del cálculo del error y de la dirección de giro del motor.

Simulación del componente ERRORPID para el control de posición:

La figura 4.11 muestra la simulación del componente ERRORPID para el caso del control de posicion. Si la referencia es de 3200 (180 grados) y el Feedback proveniente del decodificador (FB) se va aumentando el error se va reduciendo, y mientras el error sea positivo la señal “DR=0, IZ=1” será igual a cero lo que habilitara el giro a la derecha. Caso contrario cuando el error es negativo se habilitara el giro a la izquierda “IZ=0, DR=1”.

4.4 Control de velocidad.

4.4.1 Máquina finita de referencias (FSM_REF) para el control de velocidad.

La diferencia entre este componente y el componente orientado a posición son solamente las referencias. Como se calculó en el capítulo I sección 1.10.4, las referencias para la velocidad del motor son:

60rpm=160 pulsos.
70rpm=187 pulsos.
80rpm=213 pulsos.
90rpm=240 pulsos.

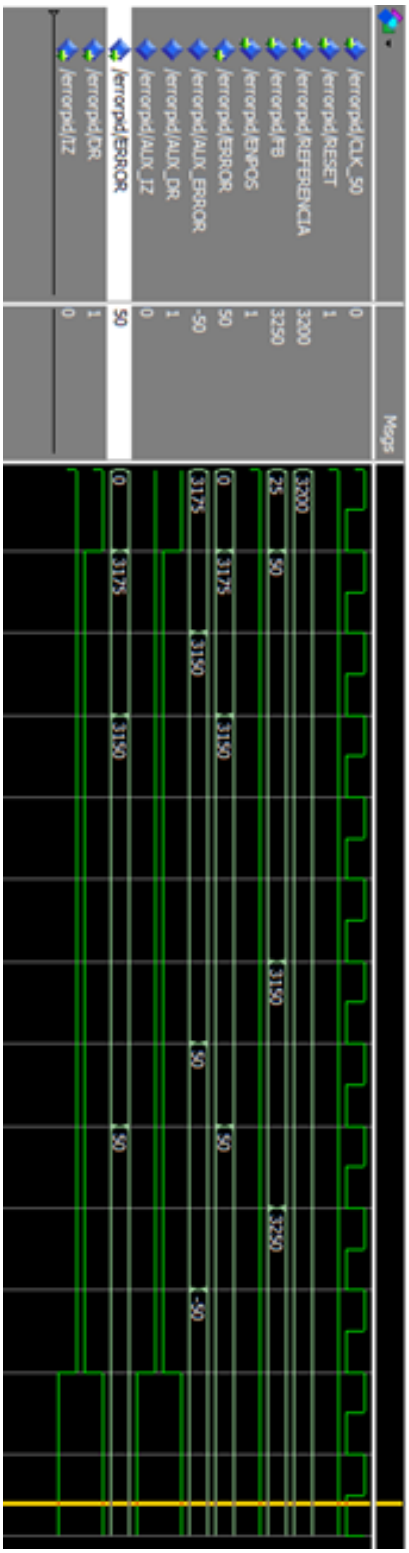


Figura 4.11. Simulación del componente ERRORPID para el control de posición.

Se opta para fines demostrativos mostrar el control solamente sobre 2 velocidades las cuales serán escogidas mediante el SW0 de la tarjeta FPGA e ingresadas al componente FSM_REF a través de la señal “”SEL”. Del mismo modo que en el caso de posición las constantes serán fijadas por medio de los SW7...SW1 y el SW8 funcionará como reset activo bajo. Para mostrar el estado actual de la máquina finita se utilizan los led LDG5…LEDG0 respectivamente los cuales cambiarán a medida que el KEY0 (CLK_FSM_REF) sea presionado.

El código de esta máquina de estados para velocidad se presenta a continuación y difiere de su homóloga de posición en las referencias de velocidad ver líneas 84-86:

Una descripción las acciones de los estados de la máquina de referencia para velocidad:

E0= estado de reset, se borra la referencia y el valor de las constantes.

E1= se lee el estado lógico del switch (switch 0) de la tarjeta a fin de obtener un binario que se hace corresponder a una referencia, ver líneas 84-85 del código 4.8.

E2= se lee el estado lógico de los switches (switches 7…1) de manera que este número binario es convertido a decimal y representara el valor de la constante KP.

E3= se lee el estado lógico de los switches (switches 7…1) de manera que este número binario es convertido a decimal y representara el valor de la constante KI.

E4= se lee el estado lógico de los switches (switches 7…2) de manera que este número binario es convertido a decimal y representara el valor de la constante KD.

E5= con el fin de que los datos sean transmitidos a los demás componentes al mismo tiempo, en el estado E5 toman valor las salidas del componente FSMREF.

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
4  USE IEEE.NUMERIC_STD.ALL;
5
6  ENTITY FSMREF IS
7
8  PORT (
9
10             RESET       :  IN STD_LOGIC;------------------------RESET DE SELECCION TIENE  SW8.
11             SEL         :  IN STD_LOGIC_VECTOR (0 DOWNTO 0);---SELECCION DE LA REFERENCIA 0,1...
12             CLK_fSM_REF :  IN STD_LOGIC;-----------------------RELOJ DE LA MAQUINA DE REFERENCIA key0
13             CONSTANTES  :  IN STD_LOGIC_VECTOR (6 DOWNTO 0);
14             REFERENCIA  :  OUT    INTEGER;---------------------REFERENCIA DE SALIDA PARA CALCULO DEL ERROR
15             ENVEL       :  OUT    STD_LOGIC;-------------------HABILITADOR VELOCIDAD.
16             KP,KI,KD    :  OUT    STD_LOGIC_VECTOR (6 DOWNTO 0);--CONSTANTES DE SALIDA
17             LED_INDICADOR: OUT    STD_LOGIC_VECTOR (5 DOWNTO 0)--INDICAN LOS ESTADOS DE LA MAQUINA
18
19 );
20
21 END FSMREF;
```

```
        PROCESS (actual_estado)
        BEGIN
           CASE actual_estado IS
                       WHEN E0 => siguiente_estado <= E1;
                       WHEN E1 => siguiente_estado <= E2;
                       WHEN E2 => siguiente_estado <= E3;
                       WHEN E3 => siguiente_estado <= E4;
                       WHEN E4 => siguiente_estado <= E5;
                       WHEN E5 => siguiente_estado <= E0;
                       WHEN OTHERS => siguiente_estado<=E0;
              END CASE ;
           END PROCESS;

PROCESS (CLK_fSM_REF,ACTUAL_ESTADO)

 BEGIN

IF (CLK_fSM_REF'event and CLK_fSM_REF='1' ) THEN
```

```
ARCHITECTURE behavioral OF FSMREF IS

 TYPE estado IS (E0,E1,E2,E3,E4,E5);-----------------------------------------5 estados para la FSM
 SIGNAL actual_estado :estado:=E0;-------------------------------------------Iniciar en el estado 0
 SIGNAL siguiente_estado : estado;
 SIGNAL AUX_KP,k_p,AUX_KI,k_i,AUX_KD,k_d : std_LOGIC_VECTOR(6 DOWNTO 0):="0000000";
 SIGNAL AUX_REF_OUT, REF_OUT : INTEGER:=0;
 SIGNAL AUX_EN0,EN0 : STD_LOGIC:= '0';
 SIGNAL LED :STD_LOGIC_VECTOR (5 DOWNTO 0):="000000";


BEGIN

   PROCESS (CLK_fSM_REF,RESET)
               BEGIN
                        IF (CLK_fSM_REF'event and CLK_fSM_REF='1') THEN
                           IF (RESET = '0') THEN
                              actual_estado<=E0;
                           ELSE
                              actual_estado<=siguiente_estado;
                           END IF;
                        END IF;
               END PROCESS;
```

```
   CASE (actual_estado) IS

                  WHEN E0=>               ------------ESTADO DE RESET

                           KP<="0000000";
                           KD<="0000000";
                           KI<="0000000";
                           ENVEL<='0';
                           REFERENCIA<=0;
                           LED<="000001";

                  WHEN E1=>

                  LED(0)<='0';

                     CASE SEL IS

                                             WHEN "0" =>AUX_REF_OUT  <= 160;-----60 RPM
                                             WHEN "1" =>AUX_REF_OUT  <= 240;------90 RPM
                                             WHEN OTHERS=>AUX_REF_OUT  <= 0   ;----0 RPM
                     END CASE;

                  LED(1)<='1';
```

```
                    WHEN E2=>

                            AUX_KP<=CONSTANTES;
                            LED(2) <='1';

                    WHEN E3=>

                            AUX_KD<=CONSTANTES;
                            LED(3) <='1';
                    WHEN E4=>

                            AUX_KI<=CONSTANTES;
                            LED(4) <='1';

                    WHEN E5=>

                       k_p<=AUX_KP;
                       k_d<=AUX_KD;
                       k_i<=AUX_KI;
                       REF_OUT<=AUX_REF_OUT;
                       AUX_EN0<='1';
                       LED(5) <='1';

            END CASE;
END IF;

               END PROCESS;

KP<=K_P;
KI<=K_I;
KD<=K_D;
ENVEL<=AUX_EN0;
REFERENCIA <= REF_OUT;
LED_INDICADOR<=LED;

END behavioral;
```

Código 4.8. Código de la máquina finita para obtener las referencias y las constantes para el controlador PID de velocidad.

4.4.2 Decodificador de velocidad (SPEED).

El tratamiento de los datos provenientes del encoder tiene diferencias obvias en cuanto a la posición. Para el caso de velocidad Cada 0.1 segundos (tiempo estipulado en la sección 1.10.4) se realiza un conteo de los pulsos provenientes del quadA del encoder con el fin de extrapolar la cantidad de pulsos para un segundo y obtener la velocidad de referencia. Para más referencia acerca del cálculo de la velocidad ver sección 1.10.4.

El código del decodificador de velocidad se presenta en el código 4.9:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SPEED IS

PORT (
        CLK_50 : IN STD_LOGIC;
        CLK_10 : IN STD_LOGIC;
        quadA : IN STD_LOGIC;
        RESET  : IN STD_LOGIC;
        VEL    : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
        RPM : INOUT INTEGER
);

END SPEED;

ARCHITECTURE behavioral OF SPEED IS
SIGNAL EN,Q1,Q2,Q3 : STD_LOGIC:='0';
SIGNAL COUNT_0 : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
SIGNAL COUNT_1 : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
SIGNAL AUX : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
SIGNAL AUX_FRPM, AUX_VEL1: INTEGER := 0;
SIGNAL D: STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
```

```
26
27  BEGIN
28
29
30  PROCESS (clk_50, quadA) BEGIN
31      IF (clk_50'event and clk_50='1') THEN
32          Q1<=quadA;
33      END IF;
34  END PROCESS;
35
36  PROCESS (clk_50,Q1) BEGIN
37      IF (clk_50'event AND clk_50='1') THEN
38         Q2<=Q1;
39      END IF;
40  END PROCESS;
41
42
43  PROCESS (clk_50,Q2) BEGIN
44      IF (clk_50'event AND clk_50='1') THEN
45          Q3<=Q2;
46        END IF;
47
48  END PROCESS;
49
```

```
50  PROCESS (clk_10) BEGIN
51  IF (clk_10'event and clk_10='1') then
52   EN<= NOT EN;
53  END IF;
54   END PROCESS;
55
56  PROCESS (Q3, CLK_50,CLK_10,EN)
57   BEGIN
58
59  IF EN = '0' THEN
60
61        IF (Q3'EVENT AND Q3='1') THEN
62           COUNT_0<=COUNT_0+1;------contador de pulsos de quadA, en 0.1segundos.
63        END IF;
64           COUNT_1<=(OTHERS=>'0');--en este punto el COUNT_1 debe ser reestablecido ya que contiene los pulsos para el periodo
65                                  --anterior del quadA
66  ELSE
67        IF (Q3'EVENT AND Q3='1') THEN
68           COUNT_1<=COUNT_1+1;
69        END IF;
70         COUNT_0<=(OTHERS=>'0');
71   END IF;
72
73   END PROCESS;
74
75   AUX<=COUNT_0+COUNT_1+1;--------------------suma de los pulsos para un periodo.  siempre hay un contador en cero.
```

```
76  PROCESS (clk_10)
77   BEGIN
78
79  IF (clk_10'event and clk_10='1') then------este FF esta sincronizado con clk_10 para que se retenga el numero de pulsos quadA contados
80        IF RESET = '0' THEN
81           D<=(OTHERS=>'0');
82        ELSE
83           D<=AUX  ;
84        END IF;
85   END IF;
86
87   END PROCESS;
88
89
90   AUX_FRPM<=TO_INTEGER (UNSIGNED (D));-------------cantidad de pulsos del quadA en un pulso de Clk_10. f= 10 hz. =0.1 segundos.
91
92   RPM <= AUX_FRPM ;--------------------------------conteos equivalentes a RPM
93
94   AUX_VEL1<= (AUX_FRPM*60*10)/(16*100);------------calculo de la velocidad en RPM
95
96   VEL <= STD_LOGIC_VECTOR(TO_UNSIGNED(AUX_VEL1,8));--conversión a binario de 8 bit para el envío por el modulo transmisor
97
98   END behavioral;
```

Código 4.9. Código decodificador de velocidad a partir de las señales en cuadratura.

Simulación del decodificador de velocidad:

La simulación muestra el momento en el que el contador “COUNT_1” ha contado 11 pulsos provenientes del “quadA” y es reestablecido a cero para que el contador “COUNT_0” inicie un nuevo conteo, todo esto sucede mientras el programa espera los flancos de subida del reloj de 10Hz (0.1 segundos de periodo definidos en sección 1.10.4).

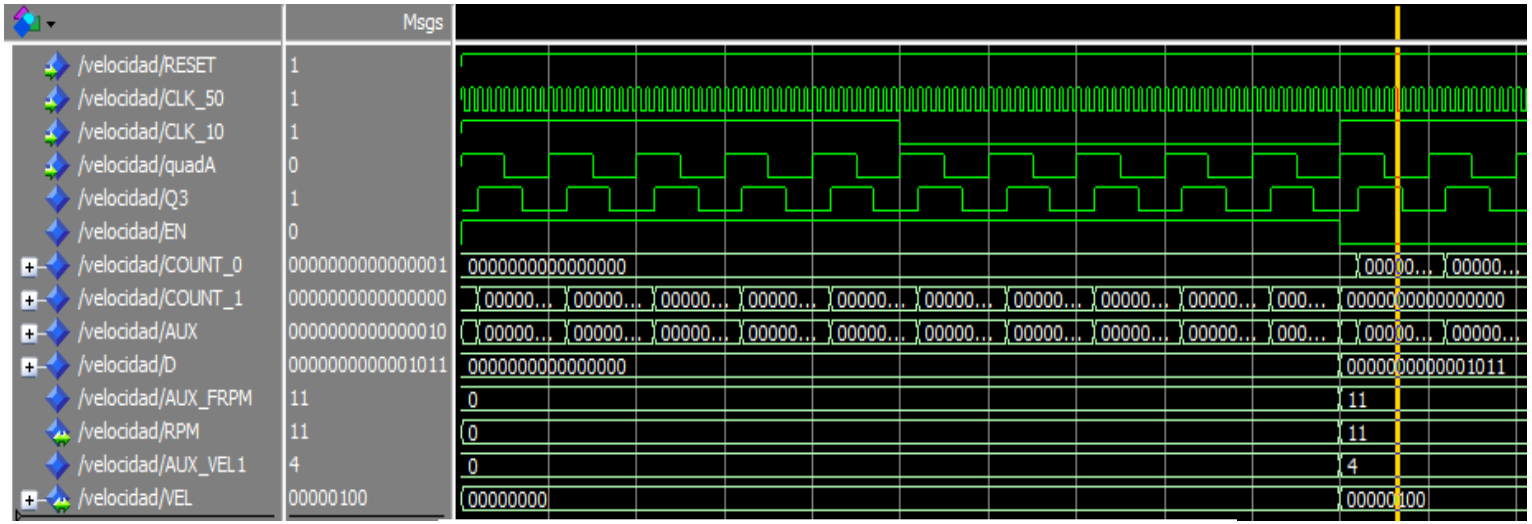


Figura 4.12. Simulación del componente decodificador de velocidad

4.4.3 Cálculo del error (ERRORPID) para control de velocidad.

El error en el caso del control de velocidad tiene la misma mecánica que el error para el control de posición, sin embargo cuando exista un error negativo no se ejecutara ninguna acción extra de corrección con las señales de dirección de giro sino que, el componente PID procesara el error negativo para reducir/aumentar la velocidad y producir una velocidad cercana a la velocidad de referencia y reducir así el error al mínimo.

Código del componente ERROR_PID:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ERRORPID IS

PORT(
        CLK_50          : IN STD_LOGIC;
        REFERENCIA      : IN INTEGER;
        RESET           : IN STD_LOGIC;
        RPM             : IN INTEGER;
        ENVEL           :  IN STD_LOGIC;
        ERROR           : INOUT INTEGER;
        IZ,DR           : OUT STD_LOGIC
);
END ERRORPID;


ARCHITECTURE behavioral OF ERRORPID IS

SIGNAL AUX_ERROR : INTEGER :=0;
SIGNAL AUX_IZ,AUX_DR :STD_LOGIC :='1';

BEGIN

PROCESS (RPM,clk_50,RESET) BEGIN

IF CLK_50'EVENT AND CLK_50 ='1' THEN

IF RESET = '0' THEN

AUX_ERROR<=0;
AUX_DR<='1';
AUX_IZ<='1';

ELSE


        IF ENVEL='1' THEN

                AUX_ERROR<=REFERENCIA - RPM;
                AUX_DR<='0';--------------giro a la derecha CW
                AUX_IZ<='1';

        END IF;

END IF;

END IF;

END PROCESS;

DR<=AUX_DR;
IZ<=AUX_IZ;

ERROR<= AUX_ERROR;


END behavioral;
```

Código 4.10. Código del componente para cálculo del error y determinación del sentido de giro.

Simulación del componente ERRORPID para el control de velocidad:

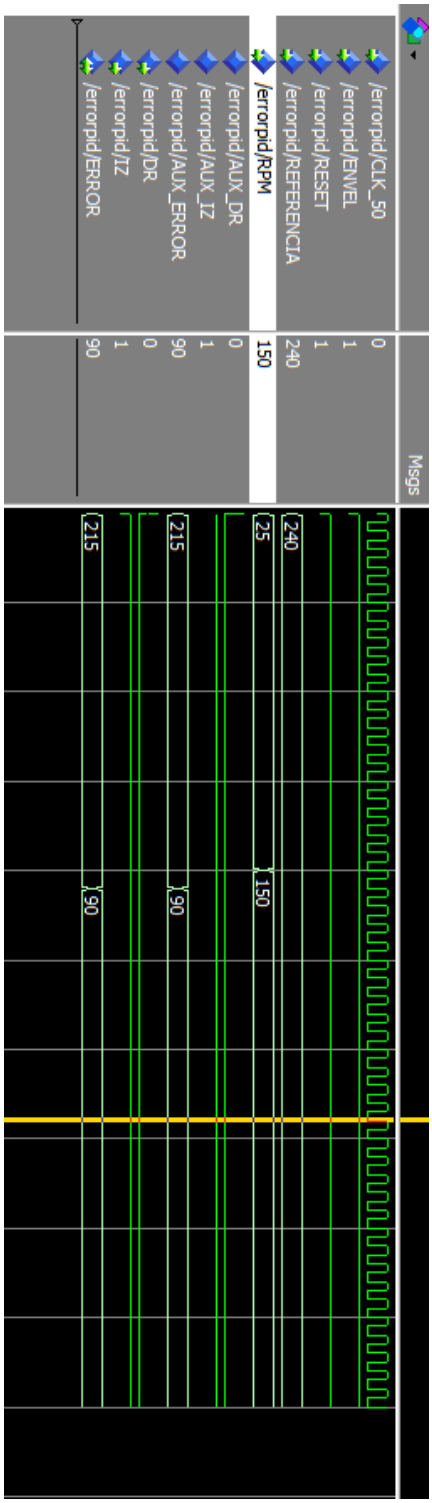


Figura 4.13. Simulación del componente ERROR PID para el caso del control de velocidad.

4.5 Resultados obtenidos

Ya que se evalúa la respuesta del sistema por medio de una entrada escalón es necesario con fines de diseño conocer el comportamiento del sistema en el tiempo. Para graficar la respuesta del sistema se utiliza el software Labview al cual se le envían los datos en tiempo real desde la tarjeta Cyclone V por medio del puerto USB usando el componente transmisor UART-USB ver sección 4.2.1. De esta forma Labview interpreta los datos enviados que ha obtenido de la tarjeta de forma serial asíncrona y grafica la posición o la velocidad del motor de corriente continua. La escalón de entrada es representada por medio de la referencia seleccionada de modo que la señal escalón original (unitaria) se ve multiplicada en magnitud por esta referencia.

La sintonización se ha llevado a cabo por el método Heurístico descrito en el capítulo I sección 1.3 el cual brinda una pauta para la selección de las constantes. La respuesta del sistema debe estar de acuerdo a requerimientos de diseño y puede ser modificada según convenga modificando las constantes, por ejemplo en la imagen 4.15 se muestran algunas características de la respuesta escalón de la figura 4.14.

Por ejemplo si se desea sintonizar el controlador para una referencia de 180 grados como primer paso se piensa usar la constante proporcional alta a fin de reducir el error rápidamente (a su máximo valor por ejemplo, 63 con 6 bit). Un valor medio de la constante integral (a la mitad de su máximo, que a su vez está limitado por el número de bit de entrada por medio de los switches, los mismos de kp-kd) a modo de conseguir un error de estado estable mínimo sin un exceso de oscilaciones. Un valor pequeño de kd con el fin de estabilizar el sistema en cuanto a sobrepaso y oscilaciones. Si se utiliza kp=63, ki=15, kd=1 se obtiene la respuesta mostrada en la figura 4.17 de la que se obtiene una respuesta con oscilación y sobrepaso, para reducir la oscilación entonces se debe reducir ki y ver cómo reacciona el sistema a este cambio, manteniendo kp = 63, reduciendo ki=8 y manteniendo kd=1 se tiene la respuesta de la figura 4.16 en la que las oscilaciones son menores. La respuesta del sistema cambiando el valor de las constantes queda sujeta a los criterios de diseño teniendo en cuenta también que el uso de constantes inapropiadas pueden hacer que el sistema caiga en inestabilidad.

4.5.1 Respuesta escalón del control de posición.

Referencia $\mathbf{220^o}$

Constantes

KP: 63

KI: 8

KD: 1

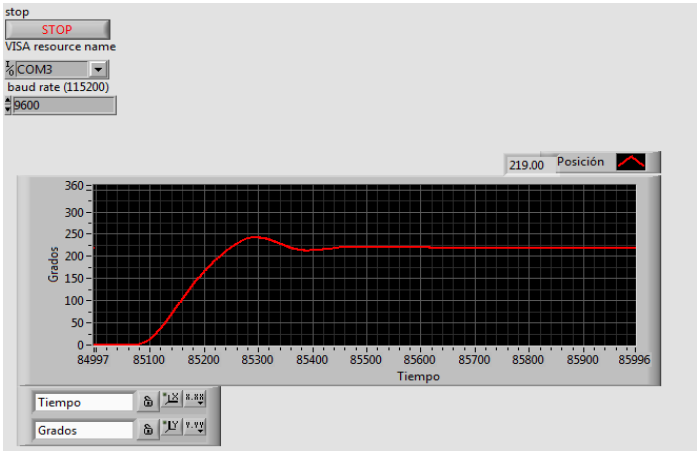


Figura 4.14. Respuesta al escalón para referencia 220 grados y constantes kp=63, ki=8 kd=1.

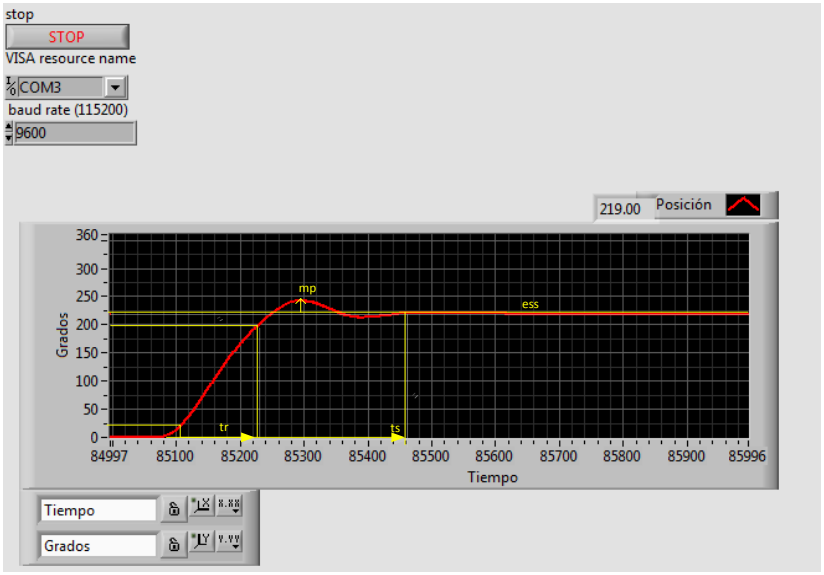


Figura 4.15. Estimación de algunos parámetros de la respuesta escalón, haciendo referencia a la fig. 4.14

Tiempo de levantamiento Tr: 120ms
Tiempo de asentamiento Ts: 380ms
Sobrepaso Máximo mp: 14^{o}
Error de estado estable ess: 1

Referencia $\mathbf{180^{o}}$

Constantes
KP: 63
KI: 8
KD: 1

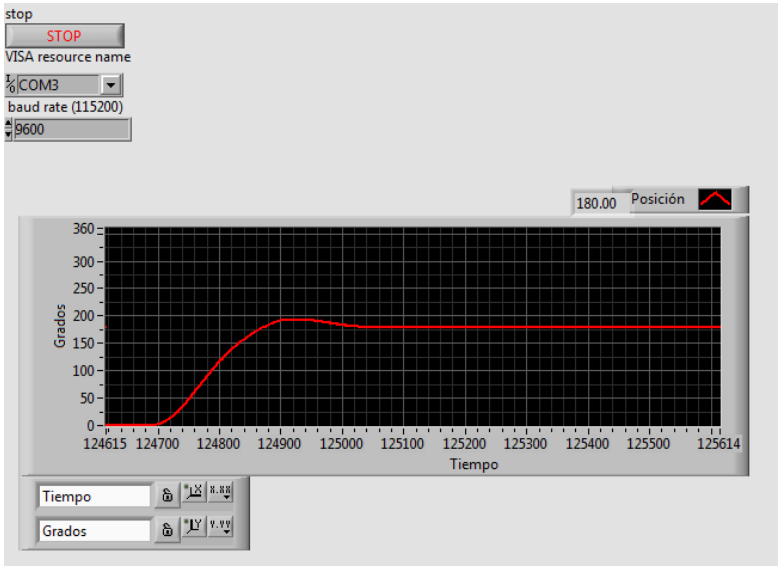


Figura 4.16. Respuesta al escalón para referencia 180 grados y constantes kp=63, ki=8 kd=1.

Constantes
KP: 63
KI: 15
KD: 1

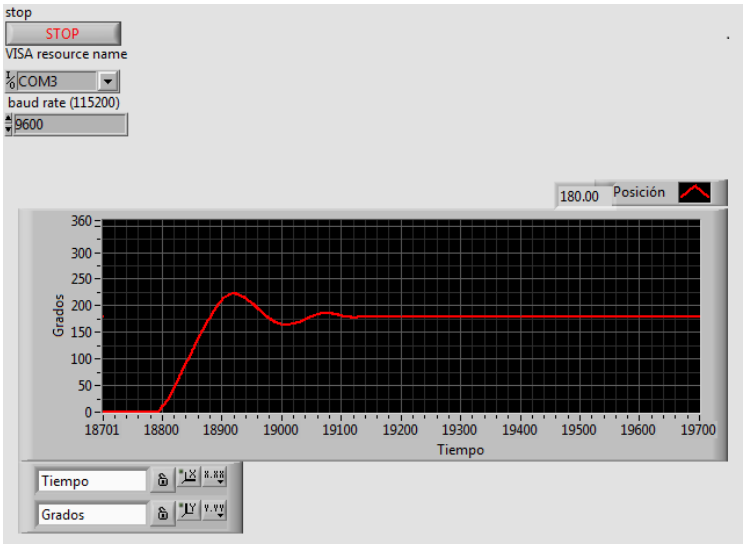


Figura 4.17. Respuesta al escalón para referencia 180 grados y constantes kp=63, ki=15, kd=1.

Referencia 90^{o}

Constantes
KP: 63
KI: 11
KD: 1

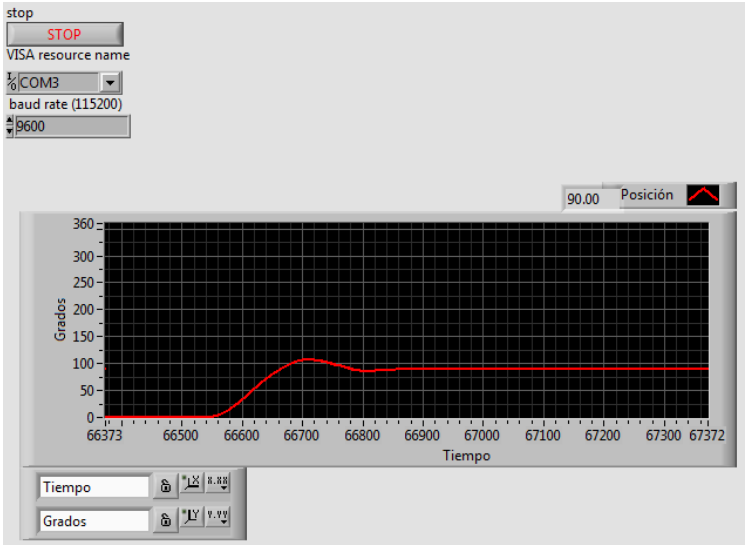


Figura 4.18. Respuesta al escalón para referencia 90 grados y constantes kp=63, ki=11, kd=1.

Referencia 45^o

Constantes

KP: 63

KI: 8

KD: 1

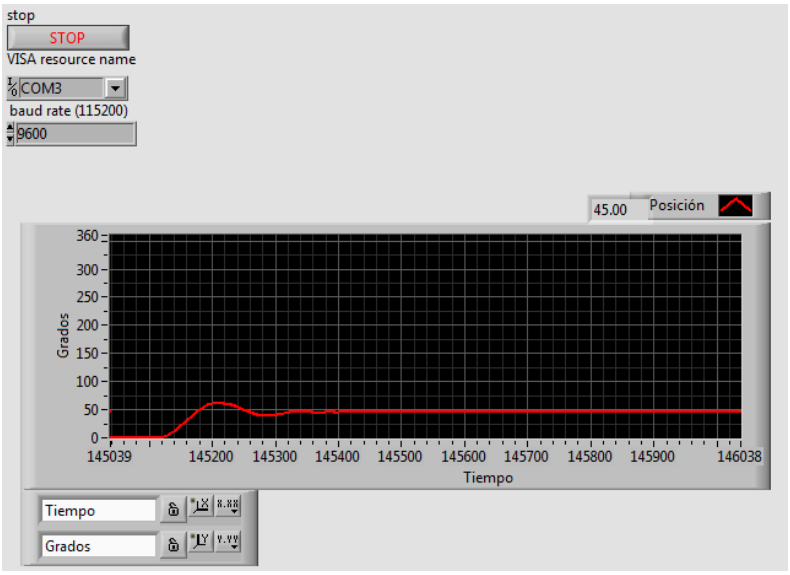


Figura 4.19. Respuesta al escalón para referencia 45 grados y constantes kp=63, ki=8, kd=1.

4.5.2 Respuesta escalón del control de velocidad.

Referencia 90 rpm

KP: 127

KI: 63

KD: 1

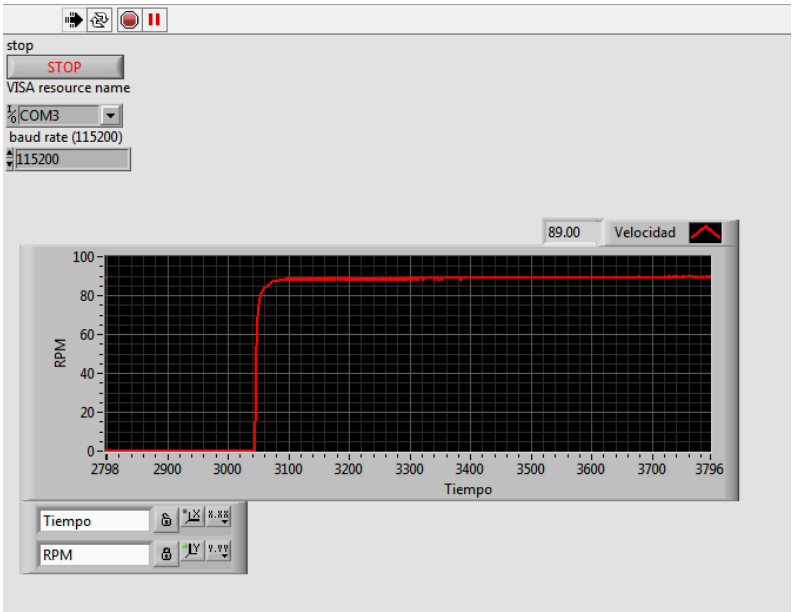


Figura 4.20. Respuesta al escalón para referencia 90rpm y constantes kp=127, ki=63, kd=1.

En otra escala:

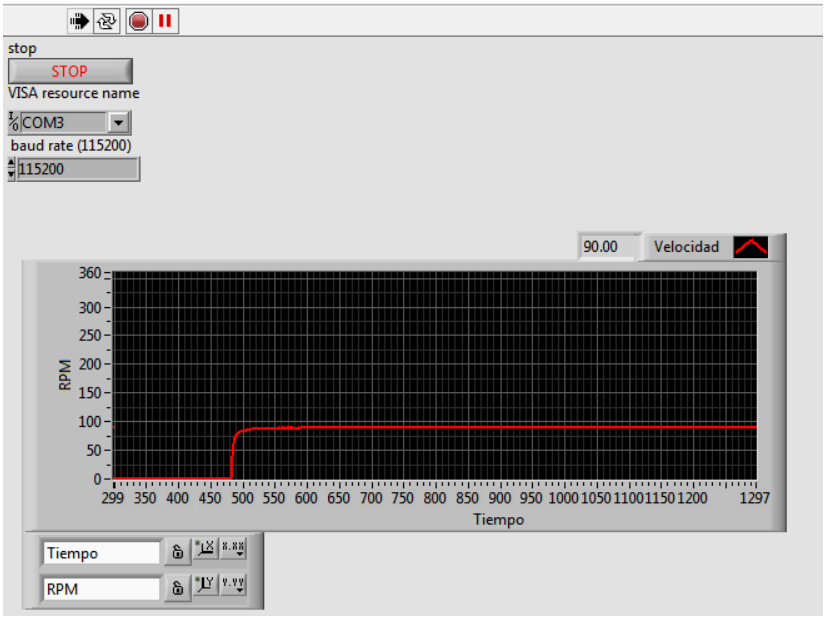


Figura 4.21. Respuesta al escalón para referencia 90rpm y constantes KP=127, KI=63 KD=1.

Referencia 70 rpm

KP: 127

KI: 63

KD: 1

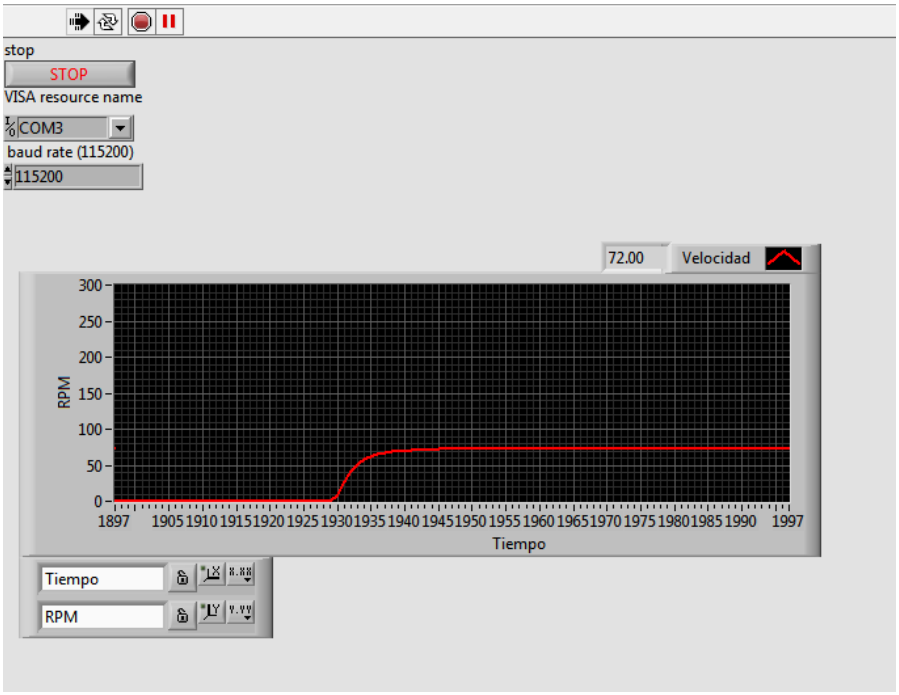


Figura 4.22. Respuesta al escalón para referencia 70rpm y constantes KP=127, KI=63 KD=1

Para la gráfica de la velocidad ya que la frecuencia de un nuevo dato es calculado cada 10Hz (ver 1.10.4) la señal TxD_INICIO del componente “TRANSMISOR” debe ajustarse para que los datos enviados no se repitan (idealmente cada 10Hz), sin embargo enviar un dato cada 10Hz puede hacer que la gráfica tarde en presentarse, aumentar la frecuencia de TxD_INICIO dara como resultado una presentación más rápida con “gradas” en la gráfica.

Para consulta del proceso de programación y uso de los controladores PID posición y velocidad para el motor de corriente continua dirigirse al anexo D.

Conclusiones.

- Los resultados obtenidos en el control de posición y velocidad del motor de corriente continua han sido satisfactorios ya que se logrado el control de posición y velocidad de este y además cambiar la respuesta del sistema en el tiempo cambiando las constantes proporcional, integral y derivativa. La respuesta del sistema queda condicionada a los parámetros de diseño requeridos y puede ser ajustada mediante el ingreso de nuevas constantes en el controlador.
- El componente controlador PID no es exclusivo del sistema de control de posición y velocidad del motor de continua, su obtención se basa en discretizar la ecuación PID para un sistema analógico. De esta forma puede usarse para otro tipo de control con el condicionante de acoplar las señales obtenidas por dispositivos (sensores, transductores) a las señales de referencia, error, actuante.
- La sintonización del controlador se realizó efectivamente mediante métodos Heurísticos, pese a que el método tiene poca referencia matemática, el conocimiento del comportamiento de la salida del sistema ante el cambio de constantes proporciona una alternativa clara más aun cuando se desconoce el modelo matemático del sistema por la falta de datos.
- Idealmente el controlador PID es para sistemas cuya función de trasferencia es de orden dos, sin embargo muchos sistemas pueden ser aproximados mediante funciones de transferencia de orden dos siendo de órdenes superiores tal es el caso del motor de corriente continua en el que se ha comprobado que el PID funciona aceptablemente.
- Implementar el controlador PID de forma discreta en la FPGA aporta facilidad en cuanto a la sintonización y el procesamiento de las señales ya que la reprogramación modifica el diseño del hardware para ajustarse al proceso requerido.
- Discretizar el PID para una FPGA implica una dificultad diferente que en otros dispositivos programables ya que las instrucciones se ejecutan de manera concurrente es necesario generar elementos lógicos que retengan los valores para cálculos posteriores.
- Existen además del efecto Windup otros problemas comunes en la realización de PID’s de carácter digital, sin embargo el efecto Windup debe ser considerado en todos estos y presenta uno de los inconvenientes más importantes a solventar.

Bibliografía

- *Kuo C, Benjamín (1996): Sistemas de Control Automático, séptima edición: Pearson Educación.*

- *Visioli, Antonio (2006): Practical PID control, Springer Science & Business Media.*

- *Basil M., Al-Hadithi (2006): Análisis y diseño de sistemas discretos de control: teoría y problemas resueltos, Visión Libros.*

- *Mendoza, Fortino (2014): Controlador estándar de movimiento multieje con base en FPGA, http://www.control-class.com/Tema_6/Slides/Tema_6_Diseno_Controladores.pdf [en línea][última consulta 12/06/1026]*

- *Narváez, Carlos A. (2008): Sintonización de controlador PID, http://documents.mx/documents/control-digital-de-velocidad-de-un-motor-dc.html [en línea][última consulta 25/05/2016]*

- *Eugenio Taccioni, Ricardo Mantz, Jorge Solsona, Pablo Puleston (2005): Controladores Basados en Estrategias PID, LEICI, Facultad de Ingeniería, UNLP: Tania Salazar & Ana Roquez http://www2.udec.cl/~vladimirfriz/Apunte-PID.pdf [en línea][última consulta 5/05/2016]*

- *Sonoli Savita, Nagabhushan Raju K. (2010): Implementation of FPGA based PID Controller for DC Motor Speed Control System. http://www.iaeng.org/publication/WCECS2010/WCECS2010_pp989-995.pdf [en línea][última consulta 8/05/2016]*

- *http://www.quees.info/que-es-hardware.html [en línea] [última consulta 26/04/2016]*

- *https://www.masadelante.com/faqs/software hardwareControl%20de%20procesos%E2%80%93FACET%E2%80%93UNT [en línea] [última consulta 18/03/2016]*

- *https://www.pololu.com/ [en línea] [última consulta 14/04/2016]*

- *Altera: C5G User Manual http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=830&PartNo=4 [en línea] [última consulta 14/04/2016]*

Referencias

- *[1.1] Kuo C, Benjamín (1996): Sistemas de Control Automático, séptima edición: Pearson Educación.*

- *[1.2] http://www.galeon.com/machver/CONTROLPROC_I/1BASICASCONTROL.pdf [en línea] [última consulta 16/05/1016]*

- *[1.3] Rodríguez Ramírez, Daniel, Teodoro Alamo, Cantarero: Diseño de controladores discretos. http://www.control-class.com/Tema_6/Slides/Tema_6_Diseno_Controladores.pdf [en línea][última consulta 10/05/2016]*

- *[1.4] Control de procesos–FACET–UNT: Métodos de sintonización de controladores PID: http://www.herrera.unt.edu.ar/controldeprocesos/tema_4/Tp4a.pdf [en línea][última consulta 15/05/2016]*

- *[1.5] Améstegui Moreno, Mauricio (2001): Apuntes de control PID, Universidad mayor de san Andrés, La paz-Bolivia http://www.info-transistor.info/biblioteca/Control%20Pid.pdf [en línea][última consulta 18/05/2016]*

- *[1.6] Mendoza, Fortino (2014): Controlador estándar de movimiento multieje con base en FPGA, http://www.control-class.com/Tema_6/Slides/Tema_6_Diseno_Controladores.pdf [en línea][última consulta 12/06/1026]*

- *[1.7]www.panamahitek.com [en línea] [última consulta 21/04/2016]*

- *[1.8] www.luisllamas.es [en línea] [última consulta 17/05/2016]*

- *[1.9] http://pcbheaven.com [en línea] [última consulta 17/05/2016]*

- *[1.10] http://www.lbaindustrial.com.mx/que-es-un-encoder/ [en línea] [última consulta 18/05/2016]*

- *[1.11] http://ceiisa.blogspot.com/2015/03/encoders.html [en línea] [última consulta 21/05/2016]*

- *[1.12] http://mechatronics.mech.northwestern.edu/design_ref/sensors/encoders.html [en línea] [última consulta 21/05/2016]*

- *[1.13] Mendoza, Fortino (2014): Controlador estándar de movimiento multieje con base en FPGA, http://www.control-class.com/Tema_6/Slides/Tema_6_Diseno_Controladores.pdf [en línea][última consulta 12/06/1026]*

- *[1.14] fpga4fun, decodificador de un encoder de cuadratura.*

- *[1.15] Gómez, Juan Carlos: Motor de Corriente Continua, Teoría de Sistemas y Señales. http://slideplayer.es/slide/1552611/ [en línea] [última consulta 20/05/2016].*

- *[1.16] http://www.el.uma.es/marin/Practica4_UART.pdf [en línea] [última consulta 22/05/2016]*

- *[1.17] http://perso.wanadoo.es/pictob/comserie.htm [en línea] [última consulta 25/05/2016]*

- *[2.1] Altera: C5G User Manual http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=830&PartNo=4 [en línea] [última consulta 14/04/2016]*

- *[2.2] https://www.pololu.com [en línea] [última consulta 12/04/2016]*

Anexo A: Laboratory Virtual Instrument Engineering Workbench (Labview).

Cuando se diseña un sistema de control o un sistema particular en el cual se está obteniendo una respuesta determinada a partir del ingreso de señales al sistema, es de interés ver este comportamiento a lo largo del tiempo o del número de muestras, el programa LABVIEW ofrece la facilidad de diseño de sistemas de adquisición de datos y su representación mediante distintas herramientas gráficas.

Labview es un potente programa diseñado por la compañía National Instrument y se basa en una programación gráfica, donde se emplea el código fuente, el lenguaje G. En principio labview se había creado solamente para sistemas operativos Macintosh, pero debido a su gran facilidad de programación y utilidad se creó para sistemas operativos Linux y Windows; se crean programas basados en diagramas de bloques que en labview se les llama Virtual Instrument (VI por sus siglas en inglés), es una buena herramienta para sistemas de hardware, pruebas de control, sistemas embebidos, entre otros. Se adapta fácilmente, ya que posee comunicación con hardware: GPIB, VXI, RS-232, RS-485, en este caso se utiliza la comunicación serial de la tarjeta FPGA cyclone V GX Starter para el envío de datos vía RS-232.

Labview presenta facilidades para el manejo de:

- *Interfaces de comunicaciones* entre las cuales se encuentran:
    - ✓ *Puerto serie*
    - ✓ *Puerto paralelo*
    - ✓ *GPIB*
    - ✓ *TCP/IP*
    - ✓ *Bluetooth*
    - ✓ *USB*
    - ✓ *Entre otros.*

- *Capacidad de interactuar con otros lenguajes y aplicaciones:*
    - ✓ *DLL*
    - ✓ *.NET*
    - ✓ *Active X*
    - ✓ *Multisim*
    - ✓ *Matlab/Simulink*
    - ✓ *AutoCAD, Solidworks, etc.*
- *Herramientas gráficas y textuales para el procesado digital de señales.*
- *Adquisición y tratamiento de imágenes.*
- *Control de movimiento.*
- *Programación de FPGAs para control o validación.*
- *Sincronización de dispositivos.*

Para la comunicación con la tarjeta FPGA, se utiliza los bloques de programación específicos para la comunicación serial, acá una breve explicación acerca de la programación utilizada en labview.

Al utilizar los bloques para la comunicación serial, se debe abrir el puerto para que exista comunicación y cuando se termine de recibir/enviar datos se debe cerrar, los bloques que hacen esta función son los siguientes:

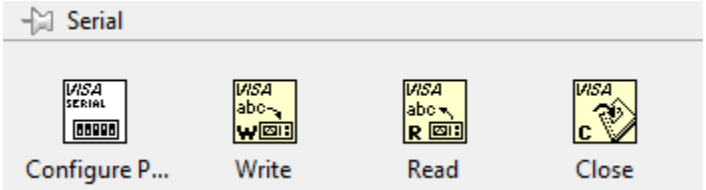


Figura A.1 Bloques para comunicación serial en labview.

Bloque Configure Serial Port.

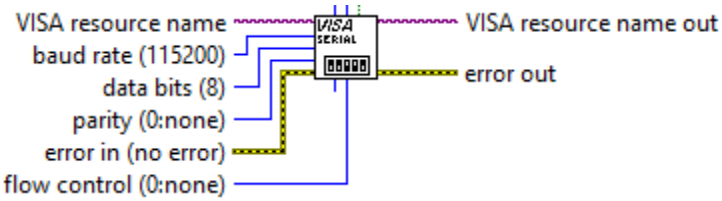


Figura A.2 Configure Serial Port.

Este bloque es el encargado de abrir el puerto para que exista comunicación serial, se utiliza de la siguiente manera:

- ✓ *VISA resource name:* acá se configura el puerto a utilizar para la comunicación serial.
- ✓ *Baud Rate:* se configura la velocidad de bits por segundo que existe la comunicación.
- ✓ *Data bits:* en esta opción se especifica en que formato se reciben los datos, se puede configurar, para recibir palabras, doble palabras y cuádruple palabras, ya sea con signo, sin signo o extendida.
- ✓ *Paridad:* se configura la paridad, en este caso no se utiliza porque no es necesario.
- ✓ *Error:* es una entrada que se utiliza para enviar una señal de error y detener el programa si ocurre algo fuera de lo normal en el programa.
- ✓ *VISA resource name out:* acá empiezan a enviarse los datos por los cuales se le ha especificado al bloque de que puerto se deben recibir.
- ✓ *Error out:* es la salida de señal de error.

Bloque Write

Figura A.3 VISA write.

El bloque para escritura se configura de la siguiente manera:

- ✓ *VISA resource name:* son los datos que provienen del puerto que se ha especificado para la comunicación de datos.
- ✓ *Write buffer:* acá se especifica el número de bytes por cada lectura.
- ✓ *Error in:* entrada de señal de error.
- ✓ *VISA resource name out:* acá empiezan a salir los datos leídos por el puerto.
- ✓ *Error out:* salida de señal de error.

Bloque Read

Figura A.4 VISA read.

VISA read se utiliza para la escritura de datos:

- ✓ *VISA resource name:* es la entrada donde se reciben los datos que se pueden escribir desde labview hacia un dispositivo electrónico.
- ✓ *Byte count:* la cantidad de datos a escribir.
- ✓ *Error in:* señal de entrada de error
- ✓ *VISA resource name:* el puerto el cual se está utilizando.
- ✓ *Read buffer:* es el buffer donde se encuentran almacenados los datos a escribir.
- ✓ *Error out:* salida se señal de error.

Bloque VISA close

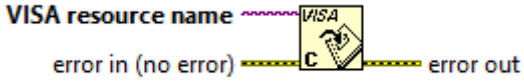


Figura A.5 VISA close.

VISA close, es el que cierra el puerto cuando finaliza el programa o interrumpe cuando sucede un error.

- ✓ *VISA resource name:* el puerto que se está utilizando.
- ✓ *Error in:* señal de entrada de error.
- ✓ *Error out:* señal de salida de error, detiene el programa en caso de error.

Bloque Lazo While

Figura A.6 Lazo while.

El Lazo while funciona como en cualquier lenguaje de programación, necesita de una condición de paro, en labview como en distintos lenguajes de programación puede poner un lazo infinito, en este caso se pone una señal de control para poder para la función de lazo while.

Bloque String to byte array

Figura A.7 Sting to byte array.

Se utiliza este bloque ya que al entrar los datos a través del puerto a la salida de este, se toman como tipo string, entonces para obtener datos correctos, se utiliza para convertir cada byte a entero sin signo.

Bloque Waveform

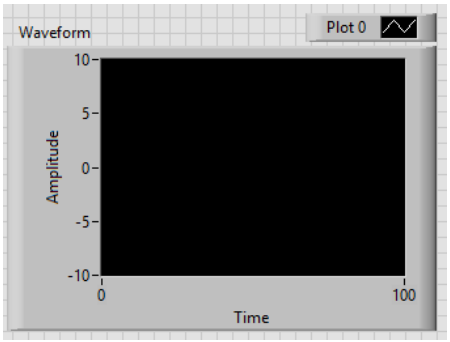


Figura A.8 Wave form chart.

El bloque wave form chart se utiliza para graficar los datos que se obtienen del puerto el cual se ha configurado para la lectura de datos desde la tarjeta FPGA.

Bloque retardo de tiempo

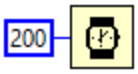


Figura A.9 retardo de tiempo.

Este bloque se utiliza para la espera de recepción de datos, la configuración del tiempo se realiza en milisegundos.

Bloque condición de paro

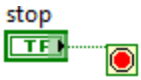


Figura A.10 condición de paro.

Botón de stop que se utiliza para la condición de paro en el programa.

Cuando ya se ha definido los bloques a utilizar, el VI queda de la siguiente manera:

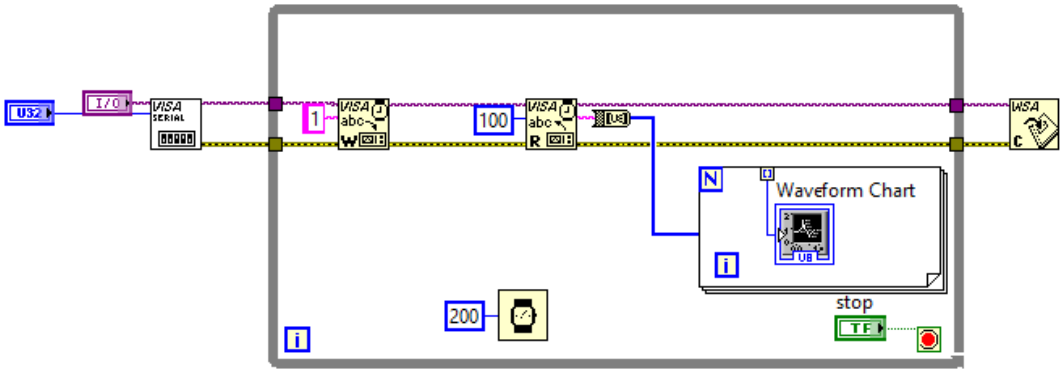


Figura A.11 Programa realizado en labview para adquisición de datos de la FPGA.

Como se reciben datos de 8 bits por cada envió de la FPGA, se le dice al bloque de lectura que se va a leer un byte por cada envío de la FPGA, luego el bloque de escritura se configura para que lea 100 bytes de datos, la salida de este bloque se convierte del formato *string* a *byte unsigned*, luego se colocan los datos en un gráfico para mostrar los datos obtenidos.
Se debe usar un retardo de 200 ms para no saturar el microprocesador de la computadora mientras se reciben datos y un control de stop con el cual se detiene el programa cuando se desee.

Anexo B: Máquinas de estado finito.

Para el diseño del controlador PID se utilizó una herramienta muy común que utilizan los programadores en diseño digital o de control, su nombre: *máquina de estado finito.* Ya que el fin de este documento no es profundizar en este tema, solo se hablara de una *breve* introducción a las máquinas de estado finito.

Las *Máquinas de estados Finitos* conocidas como *Finite State Machines (FSM)* por su traducción al inglés, sirven para realizar procesos definidos en un tiempo discreto. Reciben una entrada, hacen un proceso y entregan una salida.

Conocida también como *Autómata Finito* es una abstracción computacional que describe el comportamiento de un sistema reactivo mediante un número determinado de estados y un número determinado de transiciones entre dicho estados.
En otras palabras, es una máquina capaz de seguir una secuencia finita de pasos al introducir un conjunto de datos en ella, solo se puede leer un dato en cada paso que se realice, por tanto el número de pasos a seguir está dado por el número de datos a introducir. Cada entrada diferente genera una salida diferente, pero siempre el mismo resultado con los mismos datos de entrada.

Un autómata recibe secuencialmente una cadena de símbolos y cambia de estado por cada símbolo leído o también puede permanecer en el mismo estado. Al final de la lectura el estado del autómata indica si la cadena es aceptada o pertenece al lenguaje que describe nuestra máquina. Si al final de leer todos los símbolos de entrada de la máquina está en alguno de los estados finales entonces esa cadena es aceptada, si el estado no es final entonces la cadena no pertenece al lenguaje.

Los autómatas se componen en 5 partes, las cuales son:

$$A = \{Q, qo, F, \sum, \delta\} \qquad \text{Ecuación B.1}$$

Donde:

Q: conjunto finito de estados
qo: estado inicial donde qo pertenece a Q. Solo existe un estado inicial.
F: conjunto de estados finales F ⊆ Q.
$\sum$: Alfabeto finito de entrada.
δ: Función de transición Q × Σ → Q.

Suponer que el autómata se encuentra en el estado *qi* donde *qi* ∈ *Q*, también se tiene el símbolo *a* donde *a* ∈ *Σ*. Una entrada *a* causa que el autómata cambie del estado *qi* al estado *qk*. La función *δ*, llamada función de transición, describe este cambio de la forma *δ (qi, a)* →*qk* de esta forma obtenemos un nuevo estado. Transición es el proceso que hace un autómata al cambiar de estado. La forma más fácil de entender un autómata es mediante un diagrama de transición. Un diagrama de transición es un gráfico etiquetado con los elementos de un autómata, para este caso, pero de hecho se puede representar cualquier máquina de estados finitos. Por medio de un diagrama de transición, es la forma más común de hacerlo, por ejemplo.

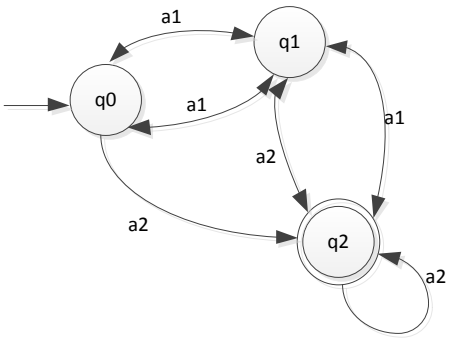


Figura B.1 Diagrama de transición.

En un diagrama de transición existe un nodo por cada estado *qi* de *Q*. Los estados finales están encerrados en un círculo doble. El estado inicial *qo* es apuntado por una flecha que no proviene de ningún otro estado. Para cada estado *qi* y un símbolo *a*, hay exactamente una y sólo una flecha que inicia en *qi* y termina en *δ(qi, a)*, es decir en *qk*, la flecha es etiquetada como *a*. Si *qk* pertenece a *F* se dice que la entrada es aceptada.

Debe haber exactamente una flecha saliendo de cada estado por cada símbolo *a0, a1, a2...an*, por tanto todos los estados tienen el mismo número de flechas saliendo de cada uno de ellos. No importa el estado ni el símbolo leído, siempre hay una transición definida.

Para describir por completo una función de transición *δ* se ocupa una tabla de transición. Las columnas se etiquetan con los símbolos de entrada, la filas son etiquetadas con los estados y en las intersecciones se colocan los nuevos estados *δ(qi, a)*, suponiendo que *qi* $\in Q$ es la columna y *a* $\in \Sigma$ la fila que lo intersecta.

	a_1	a_2
$\rightarrow q_0$	q_1	q_2
q_1	q_0	q_2
$\leftarrow q_2$	q_1	q_2

Figura B.2 tabla de transición de estados de la figura B.1

El estado inicial tiene una flecha que apunta a él, los estados finales tienen una flecha que sale de ellos y los estados que no son finales y no son el inicial no tienen flecha. En caso de que nuestro estado inicial también sea un estado final, se apunta con una flecha doble "↔".

Una tabla de transición representa una función *δ* la cual recibe un símbolo y un estado, si se desea introducir una cadena *ω* donde *ω* $\in \Sigma*$, donde $\Sigma*$ es el lenguaje universal de *Σ*, en lugar de un solo símbolo se debe usar $\delta*$ conocida como función de transición extendida que nos permite manejar una cadena dado que es una función de $Q \times \Sigma*$ y cumple con:

$\delta*(qi, a) \rightarrow \delta(qi, a)$ donde $qi \in Q$ y $a \in \Sigma$

$\delta*(qi, \varepsilon) \rightarrow \delta(qi, a)$ donde $\varepsilon \in \Sigma$ es el elemento vacío.

$\delta*(qi, aw) \rightarrow \delta*(\delta(qi, a), w)$ donde $a \in \Sigma$ y $w \in \Sigma*$

Si se evalúa δ∗ con un estado qi ∈ Q y con un símbolo a ∈ Σ se comporta de la misma forma que δ.

El primer elemento de Σ∗ generalmente es el elemento vacío ε puede ser el único elemento de nuestro lenguaje L= {ε} o se puede suponer que ε ∈(Σ∗−Σ+).

El autómata permanece en el mismo estado al introducir ε, es decir no cambia el estado, se comporta como un símbolo neutro para δ∗

El elemento ε puede ser aceptado si y solo si el estado inicial *qo* también pertenece al conjunto de estados finales *q0 ∈ F*. El autómata que solo acepta el elemento vacío es:

Figura B.3 autómata que acepta el símbolo ε.

La última propiedad de $\delta*$ define como evaluar cadenas en una forma recursiva. Se toma el primer símbolo de la cadena y se evalúa con δ, el estado resultante es evaluado con el segundo símbolo para obtener un nuevo estado que a su vez será usado con el tercer símbolo y así sucesivamente. De esta forma se pasa por los estados *δ(q0, a0), δ(δ(q0, a0), a1)...*hasta terminar de evaluar la cadena, si el autómata se encuentra en un estado final *f* se entiende que $\delta* (q0, \omega) = f$ y la cadena es aceptada.

No importa si en algún momento de la lectura de símbolos llega a un estado final, si no se ha terminado de leer la cadena, el autómata continúa recibiendo símbolos y cambiando de estado.

Formalmente se dice que un Lenguaje *L* es aceptado por un autómata *A* si y solo si $\exists \omega \in \Sigma*$ y cumple con $\delta*(q0, \omega) = f$ donde $q0 \in Q$ y $f \in F$. El lenguaje es aceptado y se compone de todos los ω de la siguiente forma:

$$L(A) = \{w \in \sum * \mid \delta * (qo, w) \in F\} \qquad \text{Ecuación B.2}$$

Anexo C: Diseño de la tarjeta de circuito impreso que controla giro del motor.

En este apartado solamente se deja en forma de referencia el diseño de la tarjeta de circuito impreso utilizado para el control de giro del motor de DC, es importante hacer énfasis en que, el diseño del circuito impreso no es único y se puede elaborar con diferentes programas de diseño de tarjetas de circuito impreso, en esta ocasión el diseño fue realizado a través del programa proteus en su versión 8.0, quedando de la siguiente manera:

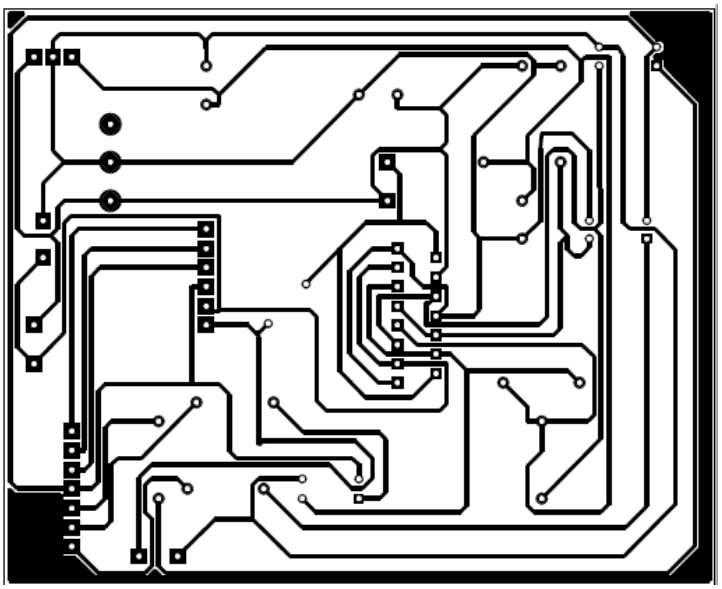


Figura C.1 diseño de la tarjeta de circuito impreso del control de giro del motor DC.

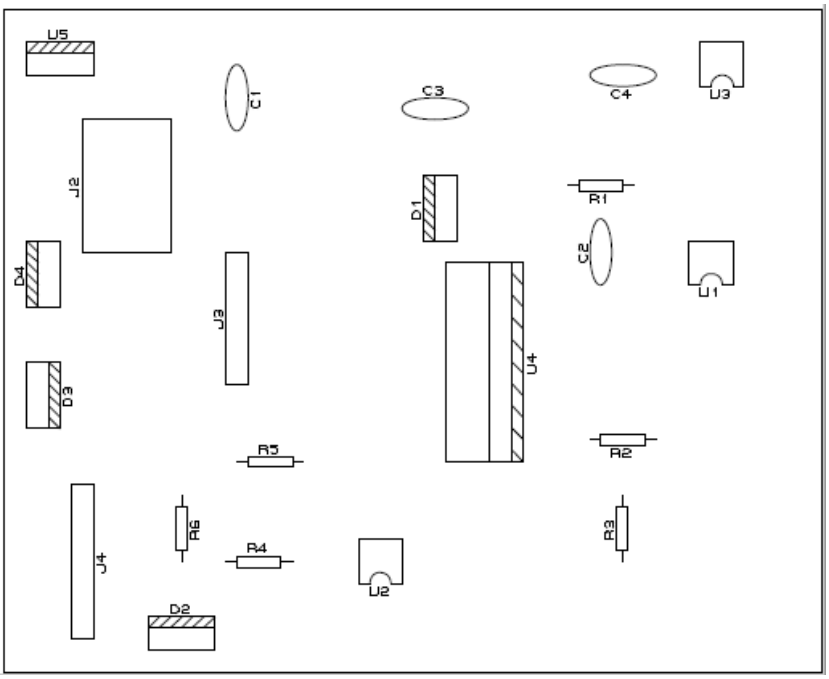


Figura C.2 ubicación de los componentes utilizados en la tarjeta de circuito impreso.

Dónde:

- U1, U2 y U3: optoacopladores TLP521
- U4: circuito integrado L298N control de motor DC
- U5: circuito integrado regulador de voltaje 7805
- D1, D2, D3 y D4: diodos semiconductores de potencia
- C1, C2, C3 y C4: condensadores cerámicos 100nF
- R1, R2 y R3: resistencias de 1KΩ
- R4, R5 y R6: resistencias de 220Ω
- J2: conector de 2 terminales
- J3: conector de 6 terminales
- J4: conector de 7 terminales

Esquemático:

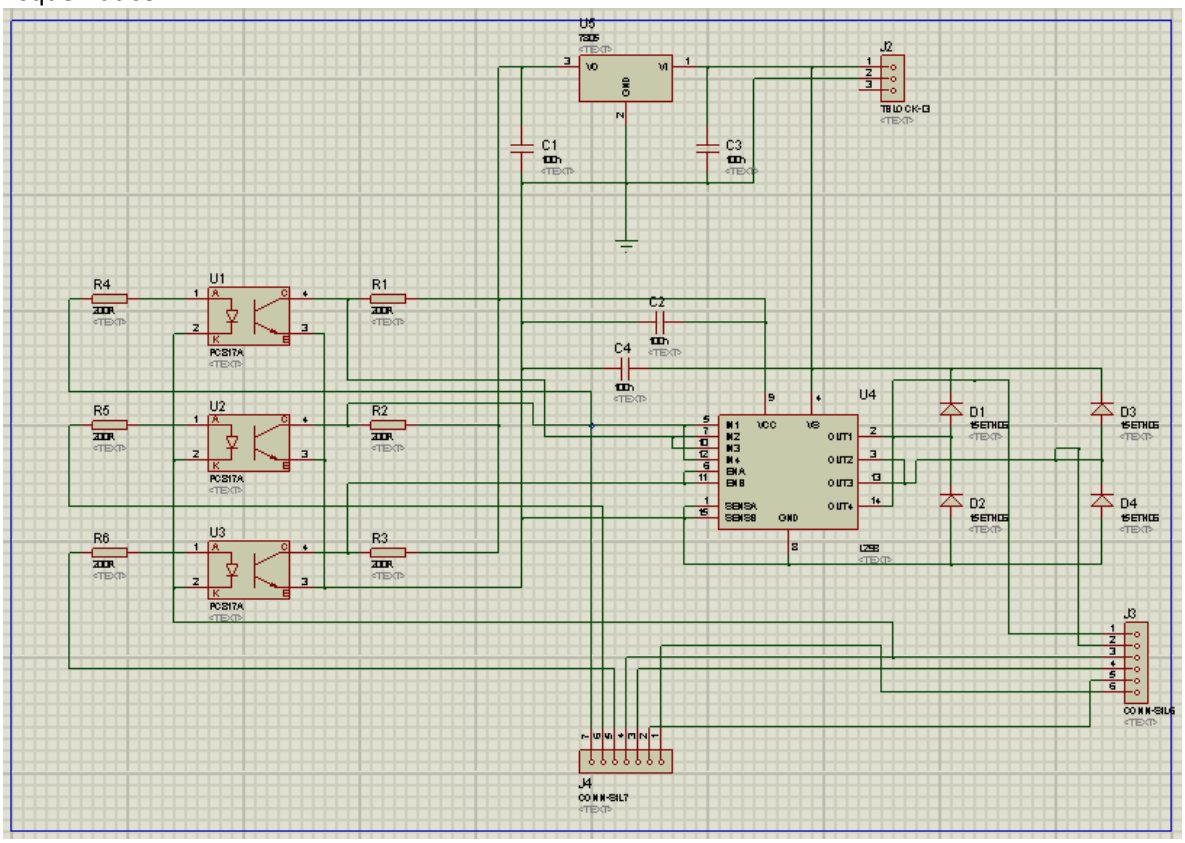


Figura C.3 esquemático del impreso en PROTEUS

Anexo D: Uso de los controladores PID posición y velocidad para el motor de corriente continua.

Tras la programación de la tarjeta cyclone V GX con los códigos de control PID de posición o control PID de velocidad se debe hacer el ingreso de las constantes y la referencia para que se realice la acción de control. Para operar los controladores PID de velocidad y PID de posición se debe tener en cuenta que la diferencia son los switch especificados para el ingreso de las constantes y referencia además del pulsador que representa el reloj de la máquina de referencias.

La tabla D.1 muestra las asignaciones correspondientes de los switches y pulsadores.

PID POSICIÓN		PID VELOCIDAD	
Asignaciones	Switches y Pulsadores	Asignaciones	Switches y Pulsadores
Reloj de la maquina finita FSMREF	KEY3 (PIN_Y16)	Reloj de la maquina finita FSRMREF	KEY0 (PIN_P11)
Referencia	SW1..SW0	Referencia	SW0
kp	SW7..SW2	kp	SW7-SW1
ki	SW7..SW2	ki	SW7-SW1
kd	SW7..SW2	kd	SW7-SW1
Reset sincrono	SW8	Reset sincrono	SW8

Tabla D.1 asignación de los switches y pulsadores para el controlador PID de posición y velocidad.

Ya que los switches asignados a las constantes kp, ki y kd del control de posición son 6 (desde el SW7 al SW2) los números ingresados podrán variar entre “0” y “63” (ya que $2^6 = 64$ combinaciones incluyendo el cero) dependiendo de la combinación binaria que se ingrese (desde 000000 hasta 111111).

Con respecto a los switches asignados al control de posición se pueden ingresar números desde “0” hasta “127” ya que se tienen 7 bit para ingreso.

Como es de esperar el SW7 es el MSB para el caso del PID posición y PID de velocidad.

La figura D.1 muestra la configuración de los pines asignados en el puerto GPIO. Es importante hacer mención que la figura D.1 no tiene la misma dirección que el puerto GPIO de la tarjeta pero si los mismos pines, la mejor manera de guiarse es medir la fuente de voltaje de 5 volt de la tarjeta y de esta forma determinar los pines “0” y “1” de esta.

El circuito impreso incluye una cincha la cual debe ser insertada como se muestra en la figura D.2, en la figura D3 se muestra una vista general de la conexión, el conexionado del motor se hace por medio de un conector del cual debe respetarse la dirección que se muestra en la figura D.4 y la conexión de la fuente externa de 12Vdc que se hace a través de una bornera.

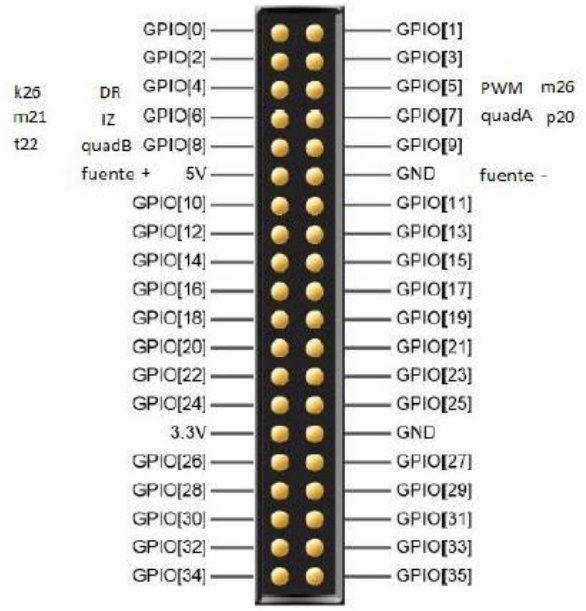


Figura D.1 Asignación de los pines al puerto GPIO

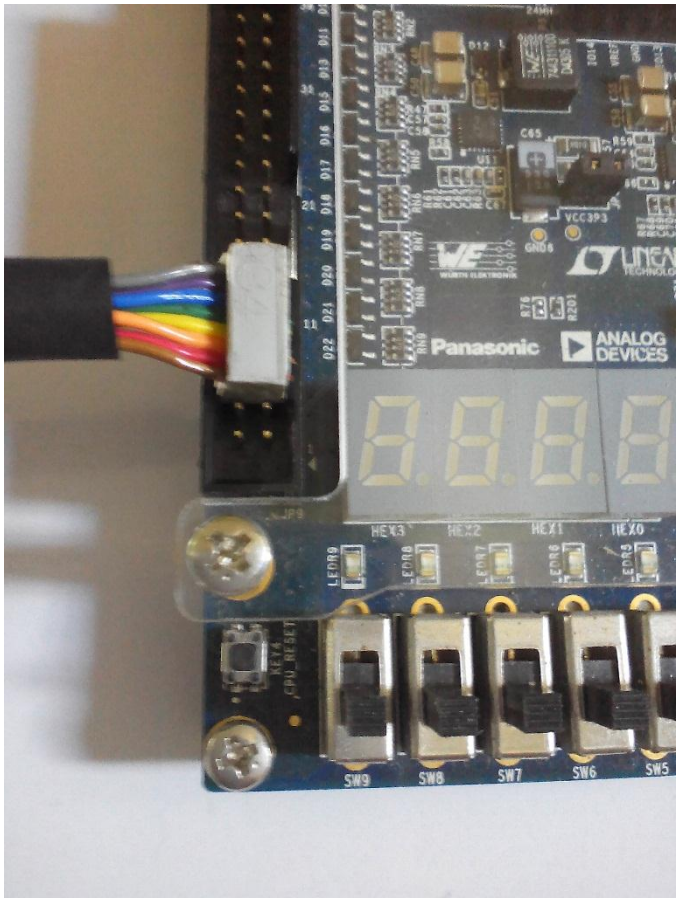


Figura D.2 conexión de la cincha a la tarjeta cyclone V GX.

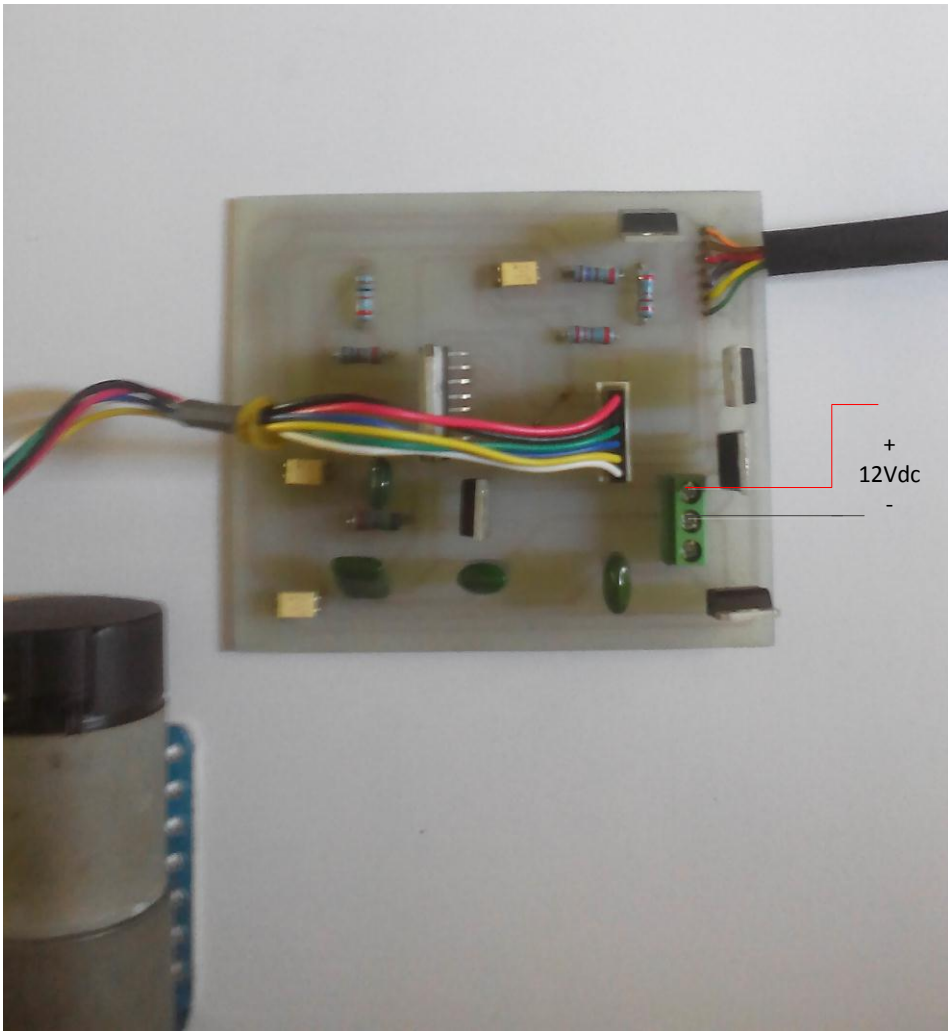


Figura D.3 Conexión del motor de continua y la fuente externa de 12Vdc

Sobre la cincha interfaz entre el circuito impreso y el GPIO se tienen las conexiones por colores, en referencia a la figura C.2, C.3 del anexo C. se describe la tabla D.2 con el fin de identificar las conexiones desde el impreso al GPIO.

Cincha interfaz ckto impreso y GPIO		
PIN7	NARANJA	DR
PIN6	CAFÉ	IZ
PIN5	ROJO	PWM
PIN4	GRIS	GND
PIN3	MORADO	(+)5vdc
PIN2	AMARILLO	quadA
PIN1	VERDE	quadB

Tabla D.2 conexión entre el circuito impreso y el GPIO.

IMPORTANTE: La conexión total se muestra en la figura D.4, en la que hay que notar que se deben de usar 2 cables USB tipo B ya que con un cable se programa la tarjeta y con el otro se transmiten los datos a la computadora, el driver necesario para que la computadora reconozca el puerto USB tipo B de la tarjeta encargado de la comunicación serial (como puerto COM) puede ser descargado de la web del fabricante:

http://www.ftdichip.com/Drivers/D2XX.htm

Para más información acerca del chip FT232R USB UART IC, descargar el cd que contiene los datasheet de los integrados que contiene la tarjeta cyclone V GX. De la página:

http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=167&No=830&PartNo=4

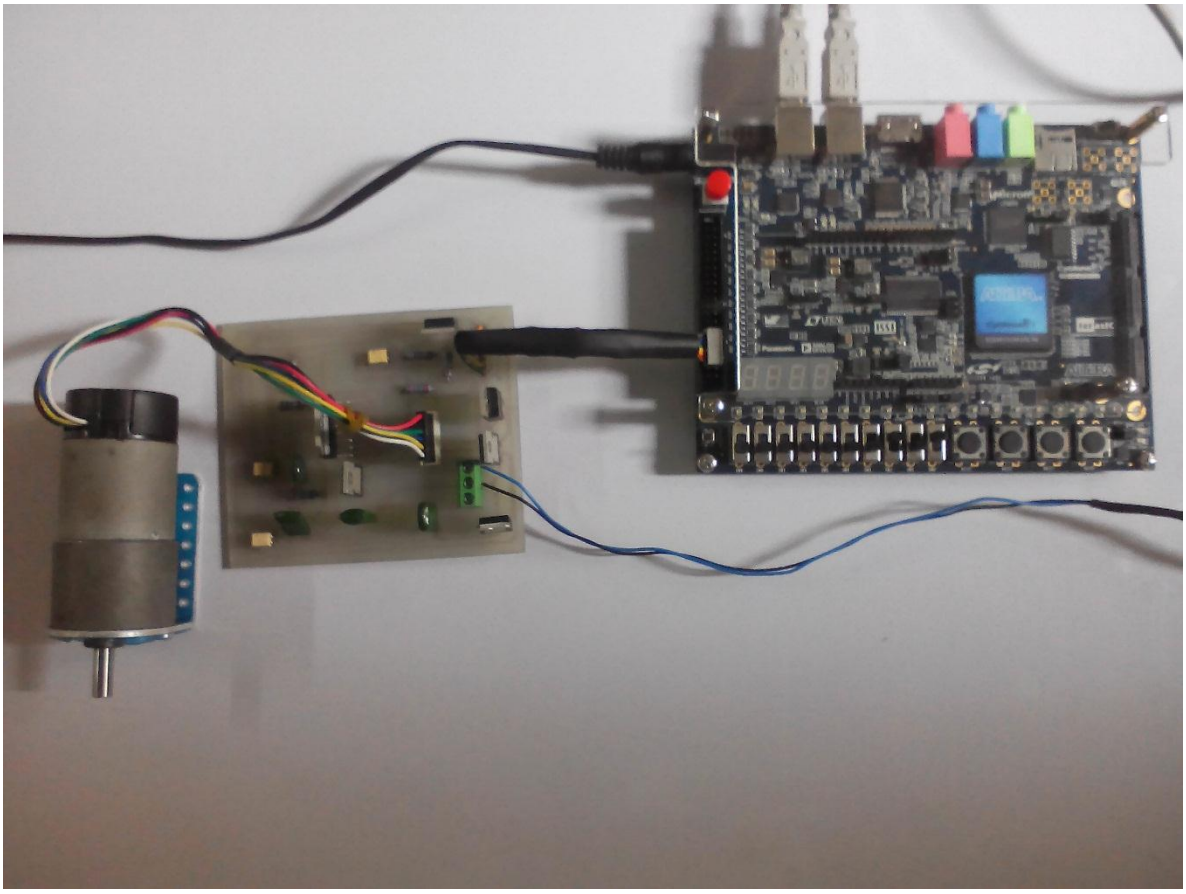


Figura D.4 conexionado total de la tarjeta cyclone V, el circuito impreso y el motor.

Habiendo corroborado las conexiones antes mencionadas, se procede a la programación de la tarjeta, ver figura D.5.

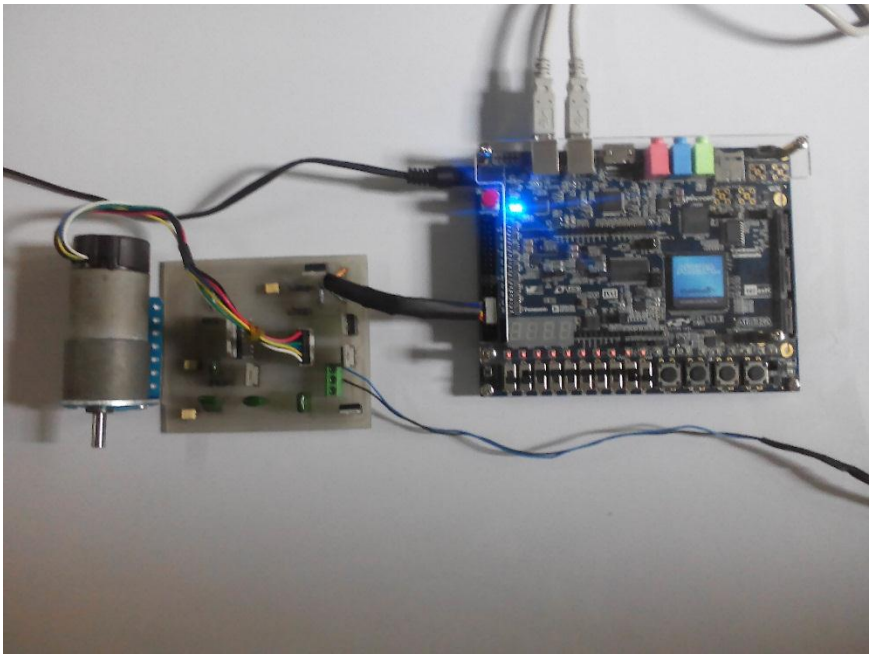


Figura D.5 tarjeta programada.

Después de programada la tarjeta está esperando que se reciba el reloj de la maquina finita FSMREF que para el caso de la posición es el KEY3 y de velocidad KEY0. En adelante se explicaran los pasos para el caso PID posición:

Luego de presionar KEY3 se entra en la maquina finita FSMREF al estado “E0” el cual es indicado mediante el LEDG0, ver figura D.6.

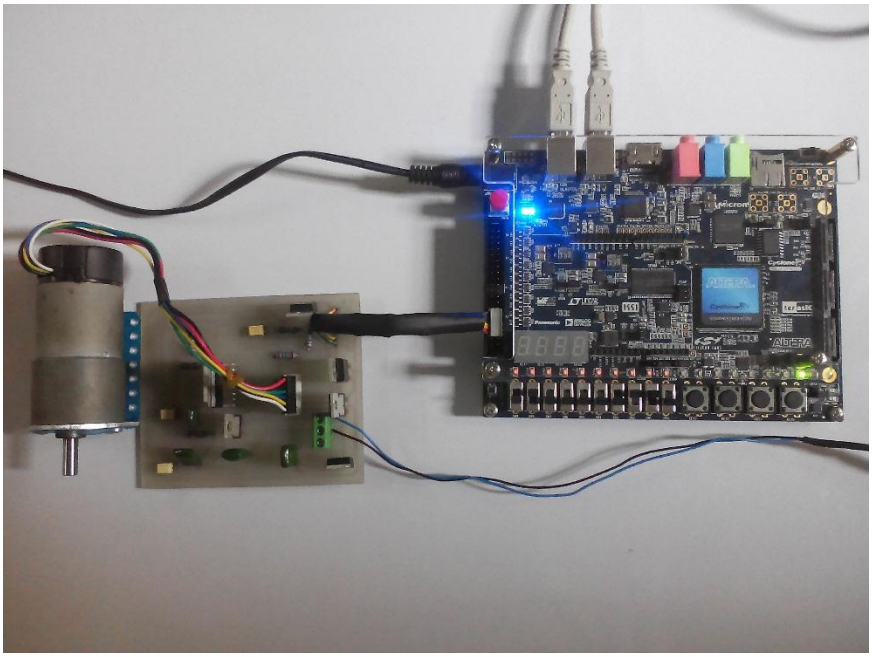


Figura D.6 máquina finita FSMREF en el estado “E0”

El siguiente paso es seleccionar la referencia. Para esto es necesario conocer las combinaciones de números binarios que representan cada una de estas, ver tabla D.3.

PID POSICIÓN		PID VELOCIDAD	
POSICIÓN	SW1..SW0	VELOCIDAD	SW0
45	"00"	70 RPM	"0"
90	"01"	90RPM	"1"
180	"10"		
220	"11"		

Tabla D.3 combinaciones binarias para la referencia.

Luego de mover los switch para establecer la referencia, se debe avanzar al estado “E1” para que esta sea guardada. Para esto hay que presionar el pulsador KEY3, la figura D.7 muestra el avance al estado “E1” y el LEDG del estado “E0” se apaga.

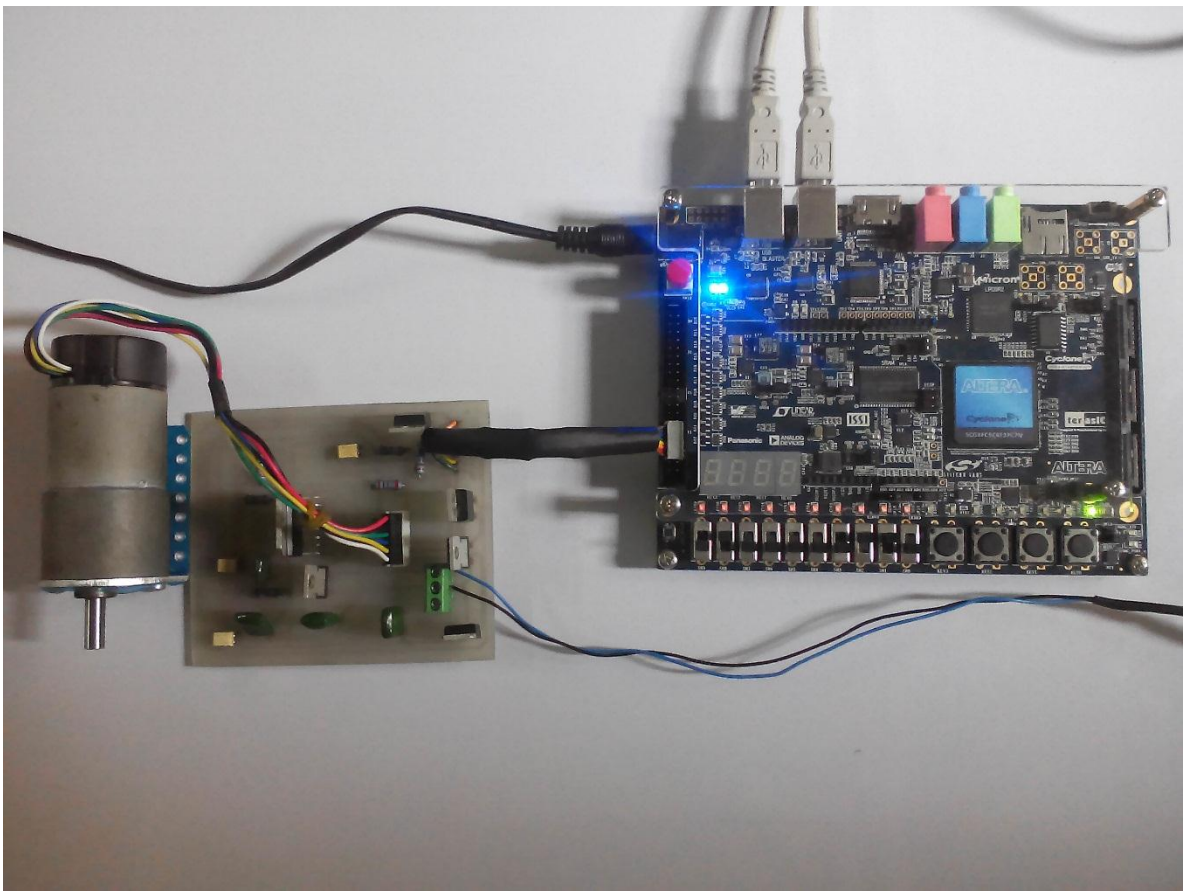


Figura D.7 máquina FSMREF en estado “E1”

En este punto ya se guardó la referencia de la posición, el siguiente estado es el “E2” el cual debe de guardar el valor de la constante proporcional kp. En este instante mientras el led indicador del estado E1 este iluminado debe de configurarse mediante los SW7..SW2 para el caso de la posición y SW7..SW1 para el caso de la velocidad el número que se va a ingresar como kp.

La figura D.8 muestra la máquina de estado en el estado “E2”.

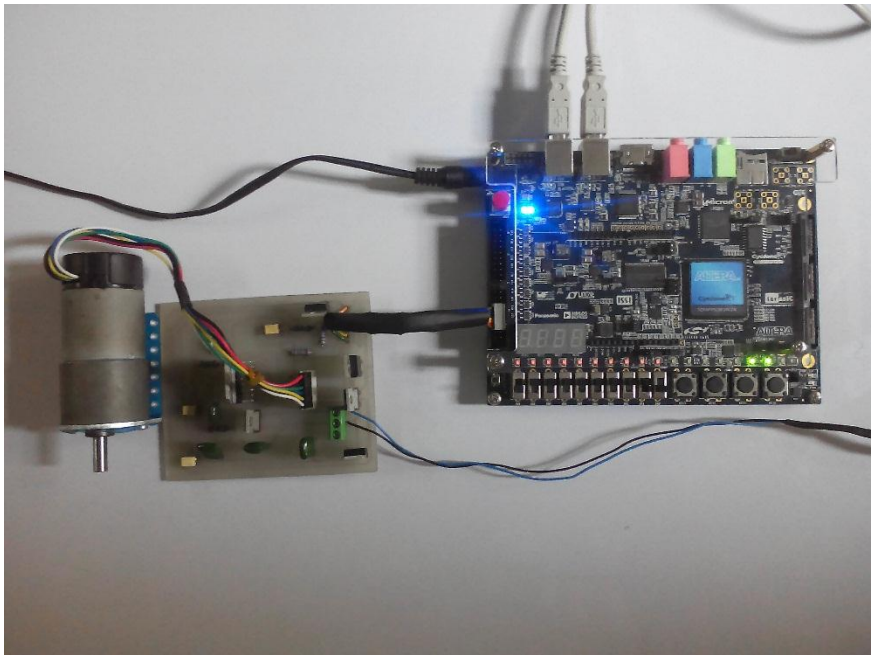


Figura D.8 máquina FSMREF en el estado “E2”.

Sigue configurar los switches de forma que se ingrese el número correspondiente a la constante integral ki. Y luego se avanza al estado “E3” para guardar este número. Ver figura D.9.

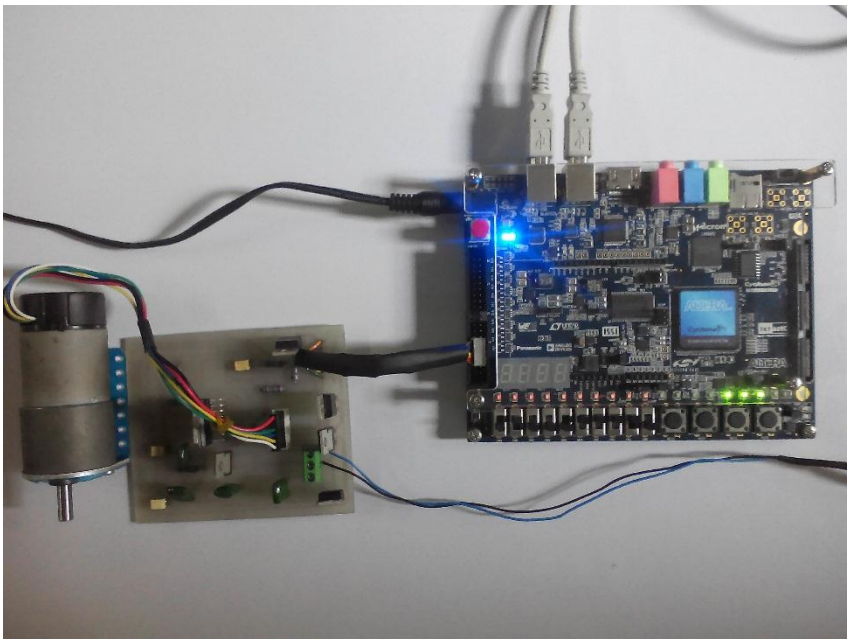


Figura D.9 máquina FSMREF en el estado “E3”.

Luego se deben configurar los switches de forma que se ingrese el número correspondiente a la constante integral kd. Y luego se avanza al estado “E4” para guardar este número. Ver figura D.10.

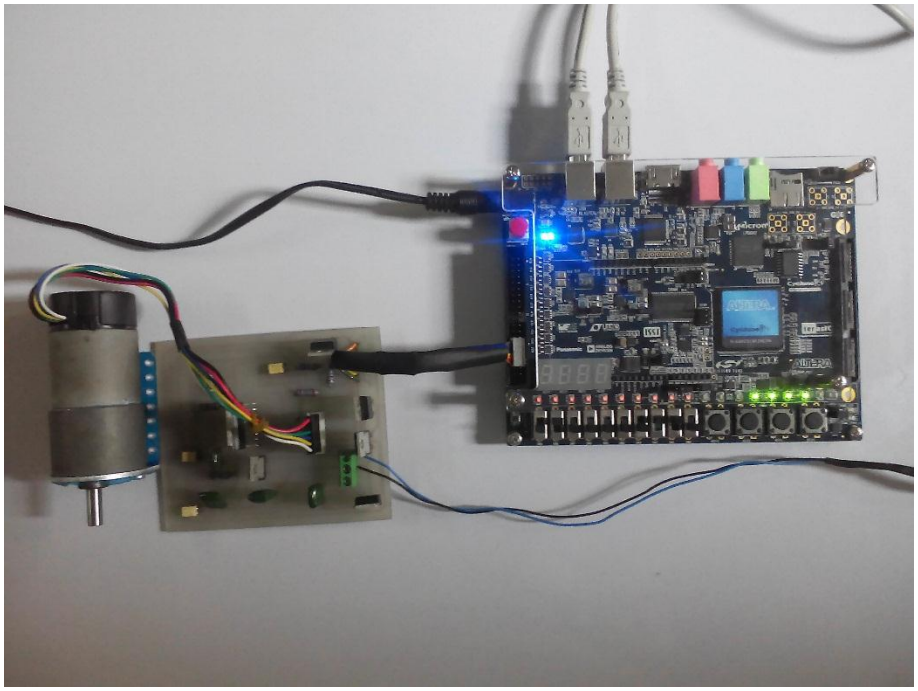


Figura D.10 máquina FSMREF en el estado “E4”.

Finalmente se transmiten los datos ingresados a los componentes respectivos cuando se ingresa al estado “E5” el cual se muestra en la figura D.11

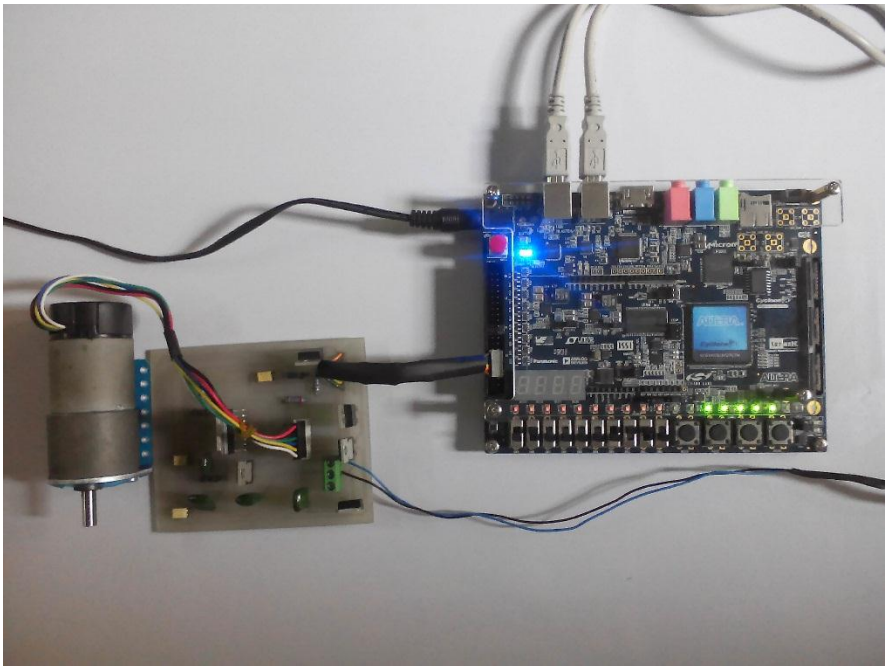


Figura D.11 máquina FSMREF en el estado “E5”.

En este momento el controlador PID ejerce el control sobre el motor para alcanzar la referencia seleccionada con las constantes ingresadas. Si se desea ingresar nueva referencia y constantes se pone el SW8 (reset) en “0” (posición baja) y se avanza normalmente en cualquier estado de la maquina FSMREF, esto hará que se borren los datos guardados ingresando al estado “E0” (recomendación: dar 2 pulsos en el estado E0 para borrar).

El proceso para utilizar LABVIEW para las gráficas de los resultados es como sigue (se deben tener los drivers VISA de labview para que se reconozca el puerto):

-Encender la tarjeta.
-Conectar la tarjeta (ambos USB).
-Programar la tarjeta.
-Abrir el programa de Labview.
-introducir el BAUD RATE (9600) y el puerto COM (aparece automático si se reconoce) con que la pc reconoce la tarjeta.
-Iniciar el programa de labview mediante la flecha “play”.
-Introducir la referencia y constantes como se ha indicado en la tarjeta cyclone V GX.
-Ver los resultados en la gráfica.

ANEXO E: Códigos de los controladores PID de posición y velocidad para el motor de corriente continua.

Código para el control de posición.

Nota: Los comentarios se muestran en verde.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;


ENTITY SELPOS IS

PORT (
      CLK_50        :IN STD_LOGIC;-------H12 CLK 50 MHZ
      quadA         :IN STD_LOGIC;-------P20          GPIO7
      quadB         :IN STD_LOGIC;-------T22          GPIO8
      CLK_fSM_REF   :IN STD_LOGIC;----- RELOJ DE LA MAQUINA DE REFERENCIA-
---
      RESET          :IN STD_LOGIC;-----RESET DE SELECCION TIENE QUE SER
UN SW.-----Y11 SW8
      SEL           :IN STD_LOGIC_VECTOR (1 DOWNTO 0);---SELECCION DE LA
REFERENCIA 00,01,10 11..,SW1,SW0
      CONSTANTES    :IN STD_LOGIC_VECTOR (6 DOWNTO 0);----
SW6,SW5,SW4,SW3..SW2, SW1
      LED_INDICADOR:OUT STD_LOGIC_VECTOR (5 DOWNTO 0);----------- INDICAN
LOS ESTADOS DE LA MAQUINA
      DR                  :OUT STD_LOGIC;             ----K26 GPIO
      IZ                  :OUT STD_LOGIC;             ----M21 GPIO
      SALIDA              :OUT STD_LOGIC;             ----M26 GPIO
      TXD                 :OUT STD_LOGIC           ----
                         );


END SELPOS;


ARCHITECTURE BEHAVIOR OF SELPOS IS

SIGNAL KP,KI,KD         : STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL GRADOS           : STD_LOGIC_VECTOR (7 DOWNTO 0):= (OTHERS=>'0');
SIGNAL REFERENCIA       : INTEGER;
SIGNAL ENPOS            : STD_LOGIC;
SIGNAL FB               : INTEGER;
SIGNAL ERROR            : INTEGER;
SIGNAL PID_OUT          : INTEGER;
SIGNAL BAUD             : STD_LOGIC;
SIGNAL TxD_INICIO       : STD_LOGIC;
SIGNAL TxD_DATO         : STD_LOGIC_VECTOR (7 DOWNTO 0):= (OTHERS => '0');
SIGNAL TxD_OCUPADO      : STD_LOGIC;
```

```
SIGNAL CLK_PID          : STD_LOGIC;
CONSTANT    FREC_BAUD   : INTEGER :=2604;
CONSTANT    FREC_TxD_INICIO : INTEGER :=125000;

COMPONENT FSMREF
PORT (

      RESET       :IN STD_LOGIC;-----RESET DE SELECCION TIENE QUE SER UN
SW.
      SEL         :IN STD_LOGIC_VECTOR (1 DOWNTO 0);---SELECCION DE LA
REFERENCIA 000,001,010...
      CLK_fSM_REF :IN STD_LOGIC;----- RELOJ DE LA MAQUINA DE REFERENCIA
      CONSTANTES  :IN STD_LOGIC_VECTOR (6 DOWNTO 0);
      REFERENCIA  :INOUT INTEGER;----------REFERENCIA DE SALIDA PARA
CALCULO DEL ERROR
      ENPOS       :INOUT STD_LOGIC;------------HABILITADOR,
POSICION,VELOCIDAD.
      KP,KI,KD    :INOUT STD_LOGIC_VECTOR (6 DOWNTO 0);---- CONSTANTES DE
SALIDA
      LED_INDICADOR:OUT  STD_LOGIC_VECTOR (5 DOWNTO 0)----------- INDICAN
LOS ESTADOS DE LA MAQUINA

);

END COMPONENT;

COMPONENT decoder
    PORT (

                clk_50: IN STD_LOGIC;
                quadA : IN STD_LOGIC;
                quadB : IN STD_LOGIC;
                RESET : IN STD_LOGIC;
                GRADOS : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                FB    : INOUT INTEGER-----------------------
REALIMENTANCION DEL ENCODER FEEDBACK
);
END COMPONENT;

COMPONENT ERRORPID

PORT(
                CLK_50                  :IN STD_LOGIC;
                REFERENCIA              :IN INTEGER;
                RESET                   :IN STD_LOGIC;
                FB                      :IN INTEGER;
                ENPOS                   :IN STD_LOGIC;
                ERROR                   :INOUT INTEGER;
                IZ,DR                   :OUT STD_LOGIC
);
END COMPONENT;

COMPONENT PID

PORT (
           CLK_PID:IN STD_LOGIC;------RELOJ DE LA MAQUINA FINITA
           ERROR  :IN INTEGER;
```

```
            KP,KI,KD :IN STD_LOGIC_VECTOR (6 DOWNTO 0);
            RESET :IN STD_LOGIC;-------------RESET AL ESTADO E1
            PID_OUT:INOUT INTEGER ----------------------UK SALIDA DEL PID
);
END COMPONENT;

COMPONENT PWM

PORT (
      clk_50      : IN  STD_LOGIC;
      PWM_IN      : IN INTEGER;------------------es la salida del PID
      SALIDA      : OUT STD_LOGIC---------------- es la señal actuante
PWM
);
END COMPONENT;

COMPONENT TRANSMISOR

PORT (
      BAUD        : IN STD_LOGIC;
      TxD_INICIO  : IN STD_LOGIC;
      TxD_DATO    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      TxD         : OUT STD_LOGIC;
      TxD_BUSY    : OUT STD_LOGIC
);

END COMPONENT;


COMPONENT GEN_FREC

PORT
(
      CLK_50      : IN STD_LOGIC;
      F_BASE      : IN INTEGER;
      FREC_OUT    : INOUT STD_LOGIC

);
END COMPONENT;


BEGIN

U1: FSMREF PORT MAP
(CLK_FSM_REF=>CLK_FSM_REF,RESET=>RESET,SEL=>SEL,CONSTANTES=>CONSTANTES,RE
FERENCIA=>REFERENCIA,ENPOS=>ENPOS,KI=>KI,KD=>KD,KP=>KP,LED_INDICADOR=>LED
_INDICADOR);

U2: DECODER PORT MAP
(CLK_50=>CLK_50,quadA=>quadA,quadB=>quadB,RESET=>RESET,FB=>FB,
GRADOS=>GRADOS);

U4: ERRORPID PORT MAP
(CLK_50=>CLK_50,REFERENCIA=>REFERENCIA,RESET=>RESET,FB=>FB,ENPOS=>ENPOS,E
RROR=>ERROR,IZ=>IZ,DR=>DR);
```

```
U5 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE => 1700,
FREC_OUT=>CLK_PID);

U6: PID PORT MAP
(CLK_PID=>CLK_PID,RESET=>RESET,ERROR=>ERROR,KP=>KP,KD=>KD,KI=>KI,PID_OUT=
>PID_OUT );

U7 : PWM PORT MAP (CLK_50=>CLK_50,PWM_IN=>PID_OUT,SALIDA=>SALIDA);

U8 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE => FREC_BAUD ,
FREC_OUT=>BAUD );

U9 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE=>  FREC_TxD_INICIO ,
FREC_OUT=>TxD_INICIO );



PROCESS (CLK_50) BEGIN
      IF CLK_50'EVENT AND CLK_50='1' THEN
            IF TxD_OCUPADO = '0' THEN
                  TxD_DATO <= GRADOS;
            END IF;
      END IF;
END PROCESS;



U10 : TRANSMISOR PORT MAP (BAUD=>BAUD, TxD_INICIO => TxD_INICIO, TxD_DATO
=> TxD_DATO , TxD => TxD, TxD_BUSY=> TxD_OCUPADO);


END BEHAVIOR;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY FSMREF IS

PORT (

RESET             :IN STD_LOGIC;-----RESET DE SELECCION TIENE QUE SER UN
SW.
SEL               :IN STD_LOGIC_VECTOR (1 DOWNTO 0);---SELECCION DE LA
REFERENCIA 000,001,010...
CLK_fSM_REF :     IN STD_LOGIC;----- RELOJ DE LA MAQUINA DE REFERENCIA
CONSTANTES  :     IN STD_LOGIC_VECTOR (6 DOWNTO 0);
REFERENCIA  :     OUT    INTEGER;----------REFERENCIA DE SALIDA PARA
CALCULO DEL ERROR
ENPOS          :  OUT    STD_LOGIC;------------HABILITADOR,
POSICION,VELOCIDAD.
KP,KI,KD    :     OUT    STD_LOGIC_VECTOR (6 DOWNTO 0);---- CONSTANTES DE
SALIDA
```

```
LED_INDICADOR:    OUT    STD_LOGIC_VECTOR (5 DOWNTO 0)---------- INDICAN
LOS ESTADOS DE LA MAQUINA

);

END FSMREF;

ARCHITECTURE behavioral OF FSMREF IS

TYPE estado IS (E0,E1,E2,E3,E4,E5);-----3 estados para la FSM
SIGNAL actual_estado :estado:=E0;-------------------------------------
------Iniciar en el estado 1
SIGNAL siguiente_estado : estado;
SIGNAL AUX_KP,AUX_KI,AUX_KD : std_LOGIC_VECTOR(6 DOWNTO 0):="0000000";
SIGNAL AUX_REF_OUT : INTEGER:=0;
SIGNAL LED :STD_LOGIC_VECTOR (5 DOWNTO 0):="000000";

BEGIN

      PROCESS (CLK_fSM_REF,RESET)
      BEGIN
            IF (CLK_fSM_REF'event and CLK_fSM_REF='1') THEN
                  IF (RESET = '0') THEN
                        actual_estado<=E0;
                  ELSE
                        actual_estado<=siguiente_estado;
                  END IF;
            END IF;
                              END PROCESS;


                  PROCESS (actual_estado)---------------El proceso se
ejecutara solo cuando la lista de sensibilidad cambie de valor, ya sea
solo un termino de muchos.
                  BEGIN
                        CASE actual_estado IS
WHEN E0 => siguiente_estado <= E1;
WHEN E1 => siguiente_estado <= E2;
WHEN E2 => siguiente_estado <= E3;
WHEN E3 => siguiente_estado <= E4;
WHEN E4 => siguiente_estado <= E5;
WHEN E5 => siguiente_estado <= E0;
WHEN OTHERS => siguiente_estado<=E0;
                                    END CASE ;
                              END PROCESS;


PROCESS (CLK_fSM_REF,ACTUAL_ESTADO)

BEGIN


IF (CLK_fSM_REF'event and CLK_fSM_REF='1' ) THEN

                  CASE (actual_estado) IS
```

```
WHEN E0=>                              ------------ESTADO DE RESET

            KP<="0000000";
            KD<="0000000";
            KI<="0000000";
            ENPOS<='0';
            REFERENCIA<=0;
            LED<="000001";

WHEN E1=>

            LED(0)<='0';

            CASE SEL IS


                     WHEN "00" =>AUX_REF_OUT <=  800;-----45 GRADOS

                     WHEN "01" =>AUX_REF_OUT <= 1600;----90 GRADOS

                     WHEN "10" =>AUX_REF_OUT <= 3200;-----180 GRADOS

                     WHEN "11" =>AUX_REF_OUT <= 3909;-----220 GRADOS

                     WHEN OTHERS=>AUX_REF_OUT <= 0;----0 GRADOS

            END CASE;


                        LED(1)<='1';

WHEN E2=>

                  AUX_KP<=CONSTANTES;
                  LED(2) <='1';

WHEN E3=>

                  AUX_KI<=CONSTANTES;
                  LED(3) <='1';
WHEN E4=>

                  AUX_KD<=CONSTANTES;
                  LED(4) <='1';

WHEN E5=>

                  KP<=AUX_KP;
                  KD<=AUX_KD;
                  KI<=AUX_KI;
                  REFERENCIA<=AUX_REF_OUT;
                  ENPOS<='1';
                  LED(5) <='1';

                        END CASE;
END IF;
```

```
END PROCESS;


LED_INDICADOR<=LED;

END behavioral;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;


ENTITY DECODER IS
    PORT ( CLK_50, quadA, quadB, RESET : IN STD_LOGIC;
           GRADOS       : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
           FB: INOUT INTEGER--------REALIMENTANCION DEL ENCODER FEEDBACK
            );
END decoder;

ARCHITECTURE Behavioral OF decoder IS

SIGNAL  Q1, Q2, Q3, Q4, Q5, Q6 : STD_LOGIC := '0' ;
SIGNAL COUNT,AUX_LED  : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
SIGNAL AUX_FB, AUX_GRADOS : INTEGER := 0;
SIGNAL DIR, CE : STD_LOGIC:='0';-------------CE = conut enable, DIR =
direccion


BEGIN


PROCESS (CLK_50, quadA, quadB) BEGIN
      IF (CLK_50'event and CLK_50='1') THEN
                  Q1<=quadA;
      END IF;
END PROCESS;


PROCESS (CLK_50, quadA, quadB) BEGIN
      IF (CLK_50'event AND CLK_50='1') THEN
                  Q2<=quadB;
      END IF;
END PROCESS;

PROCESS (CLK_50,Q1) BEGIN
      IF (CLK_50'event AND CLK_50='1') THEN
            Q3<=Q1;
      END IF;
END PROCESS;


PROCESS (CLK_50,Q2) BEGIN
      IF (CLK_50'event AND CLK_50='1') THEN
                  Q4<=Q2;
```

```
            END IF;

END PROCESS;

PROCESS (CLK_50,Q3) BEGIN
      IF (CLK_50'event AND CLK_50='1') THEN
                  Q5<=Q3;
      END IF;
END PROCESS;


PROCESS (CLK_50,Q4) BEGIN
      IF (CLK_50'event AND CLK_50='1') THEN
                  Q6<=Q4;
      END IF;
END PROCESS;


DIR <= Q3 XOR Q6;
CE  <=  Q3 XOR Q4 XOR Q5 XOR Q6;


PROCESS(CE,CLK_50) BEGIN

            IF CLK_50'EVENT AND CLK_50 = '1' THEN
                              IF RESET = '0' THEN
                                    COUNT<=(OTHERS => '0');
                              ELSE
                                    IF (CE='1') THEN --------------
CE'event AND
                                          IF DIR = '1' THEN
                                                COUNT<=COUNT+1;
                                          ELSE
                                                COUNT<=COUNT-1;
                                          END IF;
                                    END IF;
                              END IF;
            END IF;

END PROCESS;

AUX_FB <= TO_INTEGER(UNSIGNED(COUNT));
FB<=AUX_FB;

AUX_GRADOS <= (AUX_FB*360)/6400;
GRADOS <= std_logic_vector(to_unsigned(AUX_GRADOS, 8));


END Behavioral;
----------------------------------------------------------------------------
----------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ERRORPID IS
```

```
PORT(
                  CLK_50                  : IN STD_LOGIC;
                  REFERENCIA              : IN INTEGER;
                  RESET                   : IN STD_LOGIC;
                  FB                      : IN INTEGER;
                  ENPOS                   : IN STD_LOGIC;
                  ERROR                   : OUT INTEGER;
                  IZ,DR                   : OUT STD_LOGIC
);
END ERRORPID;


ARCHITECTURE behavioral OF ERRORPID IS

SIGNAL AUX_ERROR : INTEGER :=0;
SIGNAL AUX_IZ,AUX_DR :STD_LOGIC :='1';

BEGIN

PROCESS (FB,clk_50) BEGIN

      IF CLK_50'EVENT AND CLK_50 ='1' THEN

            IF RESET = '0' THEN

                  AUX_ERROR<=0;
                  AUX_DR<='1';
                  AUX_IZ<='1';

            ELSE

                  IF ENPOS='1' THEN

                  AUX_ERROR<=REFERENCIA - FB;

                  IF AUX_ERROR < 0 THEN

                  AUX_IZ<='0';--------------giro a la izquierda
                  AUX_DR<='1';--------------CCW

                        ELSIF (AUX_ERROR > 0 AND AUX_ERROR <6400 ) THEN

                  AUX_DR<='0';--------------giro a la derecha CW
                              AUX_IZ<='1';
                  ELSE
                        AUX_DR<='1';--------------DETENER MOTOR

                        AUX_IZ<='1';
                  END IF;

            END IF;


      END IF;

      END IF;
```

```
END PROCESS;

PROCESS (CLK_50) BEGIN

      IF CLK_50'EVENT AND CLK_50='1' THEN

                  IF AUX_ERROR < 0 THEN

                        ERROR<= AUX_ERROR*(-1);

                  ELSE

                        ERROR<=AUX_ERROR;

                  END IF;
      END IF;

END PROCESS;

DR<=AUX_DR;
IZ<=AUX_IZ;


END behavioral;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY PID IS

PORT (
      CLK_PID:IN STD_LOGIC;--------------RELOJ DE LA MAQUINA FINITA PID
      ERROR:IN INTEGER;------------------ERRORPID
      KP,KI,KD :IN STD_LOGIC_VECTOR (6 DOWNTO 0);--------CONSTANTES
      RESET :IN STD_LOGIC;---------------RESET ASINCRONO ACTIVO BAJO.
      PID_OUT:OUT INT--------------------SALIDA DEL PID

);
END PID;


ARCHITECTURE behavioral OF PID IS

TYPE estado IS (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15);--
---16  16 ESTADOS PARA LA FSM
SIGNAL actual_estado :estado:=E0;--------------------------------------
---INICIAR EN E0
SIGNAL siguiente_estado : estado;

SIGNAL uk,u_k,uk_1,ek,ek_1,ek_2: INTEGER:=0;
SIGNAL m_k :integer :=0;
```

```
SIGNAL AUX1,AUX2,AUX3,AUX4,AUX5,AUX6 :INTEGER:=0;----------------SENALES
PARA CALCULOS
SIGNAL A_INTEGRAL,A_I : integer:=0;-----------------------------Para
Orden de la accion integral

CONSTANT T : INTEGER:= 100;
SIGNAL AUX_KI : INTEGER:=0;
SIGNAL AUX_KP:INTEGER :=0;
SIGNAL AUX_KD : INTEGER :=0;

BEGIN
AUX_KP <= TO_INTEGER(UNSIGNED (KP));
AUX_KD <= TO_INTEGER(UNSIGNED (KD));
AUX_KI <= TO_INTEGER(UNSIGNED (KI));

      PROCESS (CLK_PID,RESET)
                              BEGIN
            IF (CLK_PID'event and CLK_PID='1') THEN
                        IF (RESET = '0') THEN
                              actual_estado<=E0;
                        ELSE
                        actual_estado<=siguiente_estado;
                        END IF;
            END IF;
      END PROCESS;


PROCESS (actual_estado)
BEGIN

CASE actual_estado IS
WHEN E0 => siguiente_estado <= E1;
WHEN E1 => siguiente_estado <= E2;
WHEN E2 => siguiente_estado <= E3;
WHEN E3 => siguiente_estado <= E4;
WHEN E4 => siguiente_estado <= E5;
WHEN E5 => siguiente_estado <= E6;
WHEN E6 => siguiente_estado <= E7;
WHEN E7 => siguiente_estado <= E8;
WHEN E8 => siguiente_estado <= E9;
WHEN E9 => siguiente_estado <= E10;
WHEN E10 => siguiente_estado <= E11;
WHEN E11 => siguiente_estado <= E12;
WHEN E12 => siguiente_estado <= E13;
WHEN E13 => siguiente_estado <= E14;
WHEN E14 => siguiente_estado <= E15;
WHEN E15 => siguiente_estado <= E1;
WHEN OTHERS => siguiente_estado<=E0;
END CASE ;
END PROCESS;


PROCESS (CLK_PID,actual_estado,error)

BEGIN

      IF (CLK_PID'event and CLK_PID='1') THEN
```

```
            CASE (actual_estado) IS

            E0=>
                                ------------ESTADO DE RESET
                        m_k  <= 0;
                        u_k  <= 0;
                        uk_1 <= 0;
                        ek <= 0;
                        ek_1 <= 0;
                        ek_2 <= 0;


      --siguiente_estado<=E1;
WHEN E1=>

ek <= error;


      --siguiente_estado<=E2;
WHEN E2=>

AUX1<=      AUX_KP*ek;


      --siguiente_estado<=E3;
WHEN E3=>

AUX2<= ((AUX_KD*T)*ek);


      --siguiente_estado<=E4;

WHEN E4=>

AUX3<= ((AUX_KI*ek_1)/(T));


      --siguiente_estado<=E5;

WHEN E5=>

AUX4<= AUX_KP*ek_1;


      --siguiente_estado<=E6;
WHEN E6=>


      AUX5<=(2*AUX_KD*T*ek_1);
```

```
      --siguiente_estado<=E7;

WHEN E7=>


      AUX6<=(AUX_KD*T)*ek_2;


      --siguiente estado<=E8;

WHEN E8=>

            A_INTEGRAL <= AUX3;


      --siguiente_estado<=E9;

WHEN E9=>
IF (A_INTEGRAL > 130000 or A_iNTEGRAL <= 0) THEN--Delimitando la salida
del controlador ResetWindup
A_I<=0;
ELSE A_I<= A_INTEGRAL;
END IF;


      --siguiente_estado<=E10;

WHEN E10=>

u_k<=uk_1+AUX1+AUX2+A_I-AUX4-AUX5+AUX6;------u_k es un auxiliar para el
calculo de la formula



      --siguiente_estado<=E11;
      WHEN E11=>

            if (u_k) > 130000 then                                    --
Delimitando la salida del controlador ResetWindup
            uk<=130000;
            elsif (u_k) <= 0 then
            uk<=0;
            else
            uk<=u_k;
            end if;

                                                                  --
siguiente_estado<=E12;

WHEN E12=>m_k<=uk;
      --------------m_k es la salida del PID

                                                              --
      siguiente_estado<=E13;
```

```
WHEN E13=>uk_1<=m_k;
      --uk_1 secuencialmente toma el valor antiguo de la ecuacion PID

                                                        --
siguiente_estado<=E14;
WHEN E14=>ek_2<=ek_1;                                                    --
error anterior menos 2<= error anterior menos 1

                                                        --
siguiente_estado<=E15;
WHEN E15=>ek_1<=ek;
      --Asignacion secuencial  error anterior menos 1 <= error anterior

                                                        --
siguiente_estado<=E1;

END CASE;
END IF;

END PROCESS;

      PID_OUT<= M_K;    ------------------------SALIDA DEL CONTROLADOR PID.

END behavioral;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY PWM IS

PORT (
            clk_50: IN  STD_LOGIC;
            PWM_IN : IN INTEGER;-------es la salida del PID
            SALIDA: OUT STD_LOGIC---- es la señal actuante PWM
            );
END PWM;

ARCHITECTURE Behavioral OF PWM IS

SIGNAL cuenta_int    : std_logic_vector(16 downto 0) := (others => '0');
SIGNAL  porcentaje : std_logic_vector(16 downto 0):=(OTHERS=>'0');

BEGIN

porcentaje <= std_logic_vector(to_unsigned(PWM_IN, 17));

P1: PROCESS (clk_50)
BEGIN

            IF (clk_50'EVENT and clk_50 = '1' ) THEN
                  IF cuenta_int = "11111111111111111" THEN
                        cuenta_int<=( OTHERS => '0');
                  ELSE
                        cuenta_int <= cuenta_int+1;
```

```
                END IF;
            END IF;

END PROCESS;

salida <= '0' WHEN (cuenta_int < porcentaje) ELSE '1';

END Behavioral;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY TRANSMISOR IS

PORT (
            BAUD        : IN STD_LOGIC;
            TxD_INICIO  : IN STD_LOGIC;
            TxD_DATO    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            TxD         : OUT STD_LOGIC;
            TxD_BUSY    : OUT STD_LOGIC
);

END TRANSMISOR;


ARCHITECTURE BEHAVIORAL OF TRANSMISOR IS

SIGNAL TxD_AUX_DATO : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL STATE : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
SIGNAL NEXT_STATE : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
SIGNAL TxD_READY : STD_LOGIC;


BEGIN

PROCESS (STATE)
BEGIN
CASE STATE IS

                WHEN "0000" => TxD_READY <= '1';
                WHEN OTHERS => TxD_READY <= '0';

END CASE;

END PROCESS;

PROCESS (STATE)
BEGIN
CASE STATE IS

                WHEN "0000" => TxD_BUSY <= '0';
                WHEN OTHERS => TxD_BUSY <= '1';

END CASE;
```

```
END PROCESS;



PROCESS (TxD_READY)
BEGIN

IF TxD_READY'EVENT AND TxD_READY ='1' THEN

TxD_AUX_DATO <= TxD_DATO;

END IF;

END PROCESS;


PROCESS (BAUD)
BEGIN
IF BAUD'EVENT AND BAUD = '1' THEN

      IF TxD_INICIO = '0' THEN
                  STATE <= "0000";
      ELSE
                  STATE<= NEXT_STATE;
      END IF;
END IF;

END PROCESS;

PROCESS (STATE)
BEGIN
CASE STATE IS

      WHEN "0000" => NEXT_STATE <= "0001";
      WHEN "0001" => NEXT_STATE <= "0010";
      WHEN "0010" => NEXT_STATE <= "0011";
      WHEN "0011" => NEXT_STATE <= "0100";
      WHEN "0100" => NEXT_STATE <= "0101";
      WHEN "0101" => NEXT_STATE <= "0110";
      WHEN "0110" => NEXT_STATE <= "0111";
      WHEN "0111" => NEXT_STATE <= "1000";
      WHEN "1000" => NEXT_STATE <= "1001";
      WHEN "1001" => NEXT_STATE <= "1010";
      WHEN "1010" => NEXT_STATE <= "1011";
      WHEN "1011" => NEXT_STATE <= "1011";
      WHEN OTHERS => NEXT_STATE <= "1011";

END CASE;

END PROCESS;


PROCESS (STATE,TXD_AUX_DATO)
BEGIN
CASE STATE IS
```

```
        WHEN "0000" => TxD  <= '1' ;
        WHEN "0001" => TxD  <= '0' ;
        WHEN "0010" => TxD  <= TxD_AUX_DATO(0);
        WHEN "0011" => TxD  <= TxD_AUX_DATO(1);
        WHEN "0100" => TxD  <= TxD_AUX_DATO(2);
        WHEN "0101" => TxD  <= TxD_AUX_DATO(3);
        WHEN "0110" => TxD  <= TxD_AUX_DATO(4);
        WHEN "0111" => TxD  <= TxD_AUX_DATO(5);
        WHEN "1000" => TxD  <= TxD_AUX_DATO(6);
        WHEN "1001" => TxD  <= TxD_AUX_DATO(7);
        WHEN "1010" => TxD  <= '1';
        WHEN "1011" => TxD  <= '1';
        WHEN OTHERS => TxD  <= '1';

END CASE;

END PROCESS;

END BEHAVIORAL;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY GEN_FREC IS
PORT
(
            CLK_50            : IN STD_LOGIC;
            F_BASE          : IN INTEGER;
            FREC_OUT    : OUT STD_LOGIC

);
END GEN_FREC;


ARCHITECTURE BEHAVIORAL OF GEN_FREC IS

SIGNAL COUNT1   : INTEGER := 0;
SIGNAL CLK2     : STD_LOGIC:='0';

BEGIN

PROCESS (CLK_50)
BEGIN

            IF CLK_50'EVENT AND CLK_50 ='1' THEN

                        IF COUNT1 >= F_BASE THEN
                              COUNT1<=0;
                              CLK2<=NOT CLK2;
                        ELSE
                              COUNT1<=COUNT1+1;
                        END IF;
```

```
            END IF;
END PROCESS;

FREC_OUT<=CLK2;

END BEHAVIORAL;
```

Código para el control de velocidad

Nota: Los comentarios se muestran en color verde.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SELVEL IS
PORT (
      CLK_50        :    IN STD_LOGIC;-------H12 CLK 50 MHZ
      quadA         :    IN STD_LOGIC;-------P20       GPIO7
      CLK_fSM_REF   :    IN STD_LOGIC;----- RELOJ DE LA MAQUINA DE
REFERENCIA----P11 KEY0
      RESET         :    IN STD_LOGIC;-----RESET DE SELECCION TIENE QUE SER
UN SW.-----Y11 SW8
      CONSTANTES    :    IN STD_LOGIC_VECTOR (6 DOWNTO 0);--PARA EL INGRESO
DE CONSTANTES
      SEL           : IN STD_LOGIC_VECTOR (0 DOWNTO 0);---SELECCION DE LA
REFERENCIA 000,001,010...SW2,SW1,SW0
LED_INDICADOR: OUT STD_LOGIC_VECTOR (5 DOWNTO 0);----------- INDICAN LOS
ESTADOS DE LA MAQUINA
TxD               : OUT STD_LOGIC;
DR                : OUT STD_LOGIC;              ----K26 GPIO 4
IZ                : OUT STD_LOGIC;              ----M26 GPIO 5
SALIDA            : OUT STD_LOGIC               ----M21 GPIO 6
                      );

END SELVEL;

ARCHITECTURE BEHAVIOR OF SELVEL IS

SIGNAL KP,KI,KD         :     STD_LOGIC_VECTOR (6 DOWNTO 0);
SIGNAL REFERENCIA       : INTEGER;
SIGNAL ENVEL            : STD_LOGIC;
SIGNAL ERROR            : INTEGER;
SIGNAL PID_OUT          : INTEGER;
SIGNAL RPM              : INTEGER;

SIGNAL BAUD             : STD_LOGIC;
SIGNAL VEL              : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL TXD_DATO         : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL TxD_INICIO       : STD_LOGIC;
SIGNAL CLK_PID          : STD_LOGIC;
SIGNAL CLK_10           : STD_LOGIC;
SIGNAL TxD_OCUPADO      : STD_LOGIC;
```

```
CONSTANT    FREC_BAUD    : INTEGER :=2604; ----9600 BAUD
CONSTANT    FREC_TxD_INICIO : INTEGER :=125000;--



COMPONENT FSMREF
PORT (

RESET       :     IN STD_LOGIC;-----RESET DE SELECCION TIENE QUE SER UN
SW.
CONSTANTES  :     IN STD_LOGIC_VECTOR (6 DOWNTO 0);
SEL         :     IN STD_LOGIC_VECTOR (0 DOWNTO 0);---SELECCION DE LA
REFERENCIA 000,001,010...
CLK_fSM_REF :     IN STD_LOGIC;----- RELOJ DE LA MAQUINA DE REFERENCIA
REFERENCIA  :     INOUT  INTEGER;----------REFERENCIA DE SALIDA PARA
CALCULO DEL ERROR
ENVEL       :     INOUT  STD_LOGIC;------------HABILITADOR,
POSICION,VELOCIDAD.
KP,KI,KD    :     INOUT  STD_LOGIC_VECTOR (6 DOWNTO 0);---- CONSTANTES DE
SALIDA
LED_INDICADOR: OUT       STD_LOGIC_VECTOR (5 DOWNTO 0)----------- INDICAN
LOS ESTADOS DE LA MAQUINA

);

END COMPONENT;

COMPONENT SPEED

PORT (
                clk_50 : IN STD_LOGIC;
                CLK_10 : IN STD_LOGIC;
                quadA  : IN STD_LOGIC;
                RESET  : IN STD_LOGIC;
                RPM    : INOUT INTEGER;
                VEL    : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0)
);
END COMPONENT;


COMPONENT ERRORPID

PORT(
                CLK_50                  :IN STD_LOGIC;
                REFERENCIA              :IN INTEGER;
                RESET                   :IN STD_LOGIC;
                RPM                     :IN INTEGER;
                ENVEL                   :IN STD_LOGIC;
                ERROR                   :INOUT INTEGER;
                IZ,DR                   :OUT STD_LOGIC
);
END COMPONENT;

COMPONENT PID

PORT (
      CLK_PID     :IN STD_LOGIC;---RELOJ DE LA MAQUINA FINITA
```

```
      ERROR       :IN INTEGER;
      KP,KI,KD    :IN STD_LOGIC_VECTOR (6 DOWNTO 0);
      RESET       :IN STD_LOGIC;----RESET AL ESTADO E1
      PID_OUT     :INOUT INTEGER ---UK SALIDA DEL PID
);
END COMPONENT;

COMPONENT PWM

PORT (
      clk_50      : IN  STD_LOGIC;
      PWM_IN      : IN INTEGER;-----------------es la salida del PID
      SALIDA      : OUT STD_LOGIC---------------- es la senial actuante
PWM
);
END COMPONENT;

COMPONENT TRANSMISOR

PORT (
      BAUD        : IN STD_LOGIC;
      TxD_INICIO  : IN STD_LOGIC;
      TxD_DATO    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
      TxD         : OUT STD_LOGIC;
      TxD_BUSY    : OUT STD_LOGIC
);

END COMPONENT;



COMPONENT GEN_FREC

PORT
(
            CLK_50            : IN STD_LOGIC;
            F_BASE          : IN INTEGER;
            FREC_OUT    : INOUT STD_LOGIC

);
END COMPONENT;



BEGIN

U1: FSMREF PORT MAP
(CLK_FSM_REF=>CLK_FSM_REF,RESET=>RESET,CONSTANTES=>CONSTANTES,SEL=>SEL,RE
FERENCIA=>REFERENCIA,ENVEL=>ENVEL,KI=>KI,KD=>KD,KP=>KP,LED_INDICADOR=>LED
_INDICADOR);


U2 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE => 2500000 ,
FREC_OUT=>CLK_10);
```

```
U3: SPEED PORT MAP (CLK_50=>CLK_50,CLK_10=>
CLK_10,quadA=>quadA,RESET=>RESET,RPM=>RPM,VEL=>VEL);

U5: ERRORPID PORT MAP
(CLK_50=>CLK_50,REFERENCIA=>REFERENCIA,RESET=>RESET,RPM=>RPM,ENVEL=>ENVEL
,ERROR=>ERROR,IZ=>IZ,DR=>DR);

U6 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE => 1700 ,
FREC_OUT=>CLK_PID);

U7: PID PORT MAP
(CLK_PID=>CLK_PID,RESET=>RESET,ERROR=>ERROR,KP=>KP,KD=>KD,KI=>KI,PID_OUT=
>PID_OUT );

U8 : PWM PORT MAP (CLK_50=>CLK_50,PWM_IN=>PID_OUT,SALIDA=>SALIDA);

U9 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE => FREC_BAUD ,
FREC_OUT=>BAUD );

U10 : GEN_FREC PORT MAP (CLK_50=>CLK_50, F_BASE=>  FREC_TxD_INICIO ,
FREC_OUT=>TxD_INICIO );



PROCESS (CLK_50) BEGIN
      IF CLK_50'EVENT AND CLK_50='1' THEN
            IF TxD_OCUPADO = '0' THEN
                  TxD_DATO <= VEL;
            END IF;
      END IF;
END PROCESS;

U11 : TRANSMISOR PORT MAP (BAUD=>BAUD, TxD_INICIO => TxD_INICIO, TxD_DATO
=> TxD_DATO, TxD => TxD, TxD_BUSY=> TxD_OCUPADO);


END BEHAVIOR;


--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY FSMREF IS
PORT (

RESET       :IN STD_LOGIC;-----RESET DE SELECCION TIENE QUE SER UN SW.
SEL         :IN STD_LOGIC_VECTOR (0 DOWNTO 0);---SELECCION DE LA
REFERENCIA 0,1...
CLK_fSM_REF :IN STD_LOGIC;----- RELOJ DE LA MAQUINA DE REFERENCIA
REFERENCIA  :OUT INTEGER;----------REFERENCIA DE SALIDA PARA CALCULO DEL
ERROR
CONSTANTES  :IN STD_LOGIC_VECTOR (6 DOWNTO 0);
ENVEL       : OUT STD_LOGIC;------------HABILITADOR, POSICION,VELOCIDAD.
```

```
KP,KI,KD     : OUT  STD_LOGIC_VECTOR (6 DOWNTO 0);---- CONSTANTES DE
SALIDA
LED_INDICADOR: OUT       STD_LOGIC_VECTOR (5 DOWNTO 0)----------- INDICAN
LOS ESTADOS DE LA MAQUINA

);

END FSMREF;

ARCHITECTURE behavioral OF FSMREF IS

TYPE estado IS (E0,E1,E2,E3,E4,E5);-----3 estados para la FSM
SIGNAL actual_estado :estado:=E0;--------------------------------------
------Iniciar en el estado 1
SIGNAL siguiente_estado : estado;
SIGNAL LED :STD_LOGIC_VECTOR (5 DOWNTO 0):="000000";
SIGNAL AUX_KP,AUX_KI,AUX_KD : std_LOGIC_VECTOR(6 DOWNTO 0):="0000000";
SIGNAL AUX_REF_OUT : INTEGER:=0;


BEGIN

PROCESS (CLK_fSM_REF,RESET)
      BEGIN
      IF (CLK_fSM_REF'event and CLK_fSM_REF='1') THEN
            IF (RESET = '0') THEN

actual_estado<=E0;
ELSE
actual_estado<=siguiente_estado;
END IF;
END IF;
END PROCESS;


PROCESS (actual_estado)---------------El proceso se ejecutara solo cuando
la lista de sensibilidad cambie de valor, ya sea solo un termino de
muchos.
BEGIN
CASE actual_estado IS
WHEN E0 => siguiente_estado <= E1;
WHEN E1 => siguiente_estado <= E2;
WHEN E2 => siguiente_estado <= E3;
WHEN E3 => siguiente_estado <= E4;
WHEN E4 => siguiente_estado <= E5;
WHEN E5 => siguiente_estado <= E0;
WHEN OTHERS => siguiente_estado<=E0;
END CASE ;
END PROCESS;


PROCESS (CLK_fSM_REF,ACTUAL_ESTADO)

BEGIN


IF (CLK_fSM_REF'event and CLK_fSM_REF='1' ) THEN
```

```
      CASE (actual_estado) IS

WHEN E0=>   ESTADO DE RESET

            KP<="0000000";
            KI<="0000000";
            KD<="0000000";
            ENVEL<='0';
            REFERENCIA<=0;
            LED<="000001";

WHEN E1=>

            LED(0)<='0';

      CASE SEL IS


             --WHEN "0" =>AUX_REF_OUT  <= 134;-----50 RPM

      --       WHEN "0" =>AUX_REF_OUT  <= 160;-----60 RPM

               WHEN "0" =>AUX_REF_OUT  <= 187;---- 70 RPM--1866

      --       WHEN "1" =>AUX_REF_OUT  <= 213;-----80 RPM

WHEN "1" =>AUX_REF_OUT  <= 240;-----90 RPM--2400

      --          WHEN "1" =>AUX_REF_OUT  <= 266;-----100 RPM

WHEN OTHERS=>AUX_REF_OUT  <= 0   ;----0 RPM
END CASE;

      LED(1)<='1';

WHEN E2=>
      AUX_KP<=CONSTANTES;
      LED(2) <='1';

WHEN E3=>

      AUX_KI<=CONSTANTES;
      LED(3) <='1';
WHEN E4=>

      AUX_KD<=CONSTANTES;
      LED(4) <='1';

WHEN E5=>

      KP<=AUX_KP;
      KI<=AUX_KI;
      KD<=AUX_KD;
      REFERENCIA<=AUX_REF_OUT;
      ENVEL<='1';
      LED(5) <='1';
```

```
      END CASE;
END IF;


END PROCESS;


LED_INDICADOR<=LED;

END behavioral;
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SPEED IS



PORT (
                  CLK_50 : IN STD_LOGIC;
                  CLK_10 : IN STD_LOGIC;
                  quadA  : IN STD_LOGIC;
                  RESET  : IN STD_LOGIC;
                  VEL    : INOUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                  RPM    : INOUT INTEGER
);

END SPEED;

ARCHITECTURE behavioral OF SPEED IS
SIGNAL EN,Q1,Q2,Q3 : STD_LOGIC:='0';
SIGNAL COUNT_0 : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
SIGNAL COUNT_1 : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
SIGNAL AUX : STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');
SIGNAL AUX_FRPM, AUX_VEL1: INTEGER := 0;
SIGNAL D: STD_LOGIC_VECTOR (15 DOWNTO 0):=(OTHERS =>'0');

BEGIN


PROCESS (clk_50, quadA) BEGIN
      IF (clk_50'event and clk_50='1') THEN
                  Q1<=quadA;
      END IF;
END PROCESS;

PROCESS (clk_50,Q1) BEGIN
      IF (clk_50'event AND clk_50='1') THEN
            Q2<=Q1;
      END IF;
END PROCESS;
```

```
PROCESS (clk_50,Q2) BEGIN
      IF (clk_50'event AND clk_50='1') THEN
                  Q3<=Q2;
            END IF;

END PROCESS;

PROCESS (clk_10) BEGIN
IF (clk_10'event and clk_10='1') then
EN<= NOT EN;
END IF;
END PROCESS;

PROCESS (Q3, CLK_50,CLK_10,EN)
BEGIN

IF EN = '0' THEN

            IF (Q3'EVENT AND Q3='1') THEN
                  COUNT_0<=COUNT_0+1;------contador de pulsos de QuadA,
estos pulsos suceden mientras sucede un periodo del clk_10 = 0.1 segundos
            END IF;
              COUNT_1<=(OTHERS=>'0');----en este punto el COUNT_1 debe
ser reestablecido ya que contiene los pulsos de QuadA para el periodo
anterior del clk_10
ELSE
            IF (Q3'EVENT AND Q3='1') THEN
                  COUNT_1<=COUNT_1+1;
            END IF;
             COUNT_0<=(OTHERS=>'0');
END IF;

END PROCESS;

AUX<=COUNT_0+COUNT_1+1;------------suma de los pulsos para un periodo.
siempre hay un contador en cero.
PROCESS (clk_10)
BEGIN

IF (clk_10'event and clk_10='1') then--------------------este FF esta
sincronizado con clk_en para que se retenga el numero de pulsos de 50MHz
CONTADOS
            IF RESET = '0' THEN
                  D<=(OTHERS=>'0');
            ELSE
                  D<=AUX  ;
            END IF;
END IF;

END PROCESS;


AUX_FRPM<=TO_INTEGER (UNSIGNED (D));--------------------------------FALTA
PROCESAR, AQUI SOLO ESTAN LA CANTIDAD DE PULSOS DE QUADA en un pulso de
Clk_10. que son 10 hz. =0.1 segundos.
```

```
RPM <= AUX_FRPM ;

AUX_VEL1<= (AUX_FRPM*60*10)/(16*100);

VEL <= STD_LOGIC_VECTOR(TO_UNSIGNED(AUX_VEL1,8));

END behavioral;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY ERRORPID IS

PORT(
                  CLK_50                  : IN STD_LOGIC;
                  REFERENCIA        : IN INTEGER;
                  RESET             : IN STD_LOGIC;
                  RPM               : IN INTEGER;
                  ENVEL             :  IN STD_LOGIC;
                  ERROR             : INOUT INTEGER;
                  IZ,DR             : OUT STD_LOGIC
);
END ERRORPID;


ARCHITECTURE behavioral OF ERRORPID IS

SIGNAL AUX_ERROR : INTEGER :=0;
SIGNAL AUX_IZ,AUX_DR :STD_LOGIC :='1';

BEGIN


PROCESS (RPM,clk_50,RESET) BEGIN

IF CLK_50'EVENT AND CLK_50 ='1' THEN

IF RESET = '0' THEN

AUX_ERROR<=0;
AUX_DR<='1';
AUX_IZ<='1';

ELSE


                  IF ENVEL='1' THEN

                    AUX_ERROR<=REFERENCIA - RPM;
                    AUX_DR<='0';--------------giro a la derecha CW
                  AUX_IZ<='1';
```

```
                END IF;

END IF;

END IF;

END PROCESS;

DR<=AUX_DR;
IZ<=AUX_IZ;


ERROR<= AUX_ERROR;


END behavioral;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY PID IS

PORT (
      CLK_PID:IN STD_LOGIC;--------------RELOJ DE LA MAQUINA FINITA PID
      ERROR  :IN INTEGER;-------------------------------------------
ERRORPID
      KP,KI,KD:IN STD_LOGIC_VECTOR (6 DOWNTO 0);-------------CONSTANTES
      RESET :IN STD_LOGIC;----------------RESET ASINCRONO ACTIVO BAJO.
      PID_OUT:OUT INTEGER------------------SALIDA DEL PID

);
END PID;


ARCHITECTURE behavioral OF PID IS

TYPE estado IS (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9,E10,E11,E12,E13,E14,E15);--
---16  16 ESTADOS PARA LA FSM
SIGNAL actual_estado :estado:=E0;-------------------------------------
---INICIAR EN E0
SIGNAL siguiente_estado : estado;

SIGNAL uk,u_k,uk_1,ek,ek_1,ek_2: INTEGER:=0;
SIGNAL m_k :integer :=0;
SIGNAL AUX1,AUX2,AUX3,AUX4,AUX5,AUX6 :INTEGER:=0;----------------SENALES
PARA CALCULOS
SIGNAL A_INTEGRAL,A_I : integer:=0;-----------------------------Para
Orden de la accion integral

CONSTANT T : INTEGER:= 100;
SIGNAL AUX_KI : INTEGER:=0;
SIGNAL AUX_KP:INTEGER :=0;
SIGNAL AUX_KD : INTEGER :=0;
```

```
BEGIN
AUX_KP <= TO_INTEGER(UNSIGNED (KP));
AUX_KD <= TO_INTEGER(UNSIGNED (KD));
AUX_KI <= TO_INTEGER(UNSIGNED (KI));

      PROCESS (CLK_PID,RESET)
            BEGIN
                  IF (CLK_PID'event and CLK_PID='1') THEN
                        IF (RESET = '0') THEN
                              actual_estado<=E0;
                        ELSE
                              actual_estado<=siguiente_estado;
                        END IF;
                  END IF;
      END PROCESS;


PROCESS (actual_estado)
BEGIN

CASE actual_estado IS
WHEN E0 => siguiente_estado <= E1;
WHEN E1 => siguiente_estado <= E2;
WHEN E2 => siguiente_estado <= E3;
WHEN E3 => siguiente_estado <= E4;
WHEN E4 => siguiente_estado <= E5;
WHEN E5 => siguiente_estado <= E6;
WHEN E6 => siguiente_estado <= E7;
WHEN E7 => siguiente_estado <= E8;
WHEN E8 => siguiente_estado <= E9;
WHEN E9 => siguiente_estado <= E10;
WHEN E10 => siguiente_estado <= E11;
WHEN E11 => siguiente_estado <= E12;
WHEN E12 => siguiente_estado <= E13;
WHEN E13 => siguiente_estado <= E14;
WHEN E14 => siguiente_estado <= E15;
WHEN E15 => siguiente_estado <= E1;
WHEN OTHERS => siguiente_estado<=E0;
END CASE ;

END PROCESS;


PROCESS (CLK_PID,actual_estado,error)

      BEGIN

      IF (CLK_PID'event and CLK_PID='1') THEN

      CASE (actual_estado) IS

      WHEN E0=>
                        ------------ESTADO DE RESET
                  m_k  <= 0;
                  u_k  <= 0;
                  uk_1 <= 0;
```

```
            ek <= 0;
            ek_1 <= 0;
            ek_2 <= 0;

--siguiente estado<=E1;

WHEN E1=>

            ek <= error;

--siguiente_estado<=E2;

WHEN E2=>

            AUX1<=      AUX_KP*ek;

--siguiente_estado<=E3;

WHEN E3=>

            AUX2<= ((AUX_KD*T)*ek);

--siguiente_estado<=E4;

WHEN E4=>

            AUX3<= ((AUX_KI*ek_1)/(T));

--siguiente_estado<=E5;

WHEN E5=>

            AUX4<= AUX_KP*ek_1;

--siguiente_estado<=E6;

WHEN E6=>

             AUX5<=(2*AUX_KD*T*ek_1);

--siguiente_estado<=E7;

WHEN E7=>

             AUX6<=(AUX_KD*T)*ek_2;

--siguiente_estado<=E8;
```

```
      WHEN E8=>

                  A_INTEGRAL <= AUX3;


      --siguiente_estado<=E9;

      WHEN E9=>

      IF (A_INTEGRAL > 130000 or A_iNTEGRAL <= 0) THEN--Delimitando la
salida del controlador ResetWindup
                  A_I<=0;
      ELSE A_I<= A_INTEGRAL;
      END IF;


      --siguiente_estado<=E10;

      WHEN E10=>

            u_k<=uk_1+AUX1+AUX2+A_I-AUX4-AUX5+AUX6;------u_k es un
auxiliar para el calculo de la formula


      --siguiente_estado<=E11;
      WHEN E11=>

            IF (u_k) > 130000 then                                    --
Delimitando la salida del controlador ResetWindup
             uk<=130000;
            elsif (u_k) <= 0 then
                  uk<=0;
            else
                  uk<=u_k;
            end if;

                                                                  --
siguiente_estado<=E12;

      WHEN E12=>m_k<=uk;
            --------------m_k es la salida del PID

                                                              --
      siguiente_estado<=E13;
      WHEN E13=>uk_1<=m_k;
            --uk_1 secuencialmente toma el valor antiguo de la ecuacion
PID

            --siguiente_estado<=E14;
      WHEN E14=>ek_2<=ek_1;
      --error anterior menos 2<= error anterior menos

      --siguiente_estado<=E15;
      WHEN E15=>ek_1<=ek;
            --Asignacion secuencial  error anterior menos 1 <= error
anterior
```

```
                                                                        --
siguiente_estado<=E1;

END CASE;
END IF;

END PROCESS;

      PID_OUT<= M_K;    ----------------------SALIDA DEL CONTROLADOR PID.

END behavioral;
--------------------------------------------------------------------------
--------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY PWM IS

PORT (
            clk_50            : IN  STD_LOGIC;
            PWM_IN            : IN INTEGER;-----------------es la salida
del PID
             SALIDA           : OUT STD_LOGIC---------------- es la senial
actuante PWM
            );

END PWM;

ARCHITECTURE Behavioral OF PWM IS

SIGNAL cuenta_int    : std_logic_vector(16 downto 0) := (others => '0');
SIGNAL  porcentaje : std_logic_vector(16 downto 0):=(OTHERS=>'0');

BEGIN

porcentaje <= std_logic_vector(to_unsigned(PWM_IN, 17));


P1: PROCESS (clk_50)
BEGIN

            IF (clk_50'EVENT and clk_50 = '1' ) THEN
                  IF cuenta_int = "11111111111111111" THEN---------------
--------99%
                        cuenta_int<=( OTHERS => '0');
                  ELSE
                        cuenta_int <= cuenta_int+1;----------------------
--
                  END IF;
            END IF;

END PROCESS;

salida <= '0' WHEN (cuenta_int < porcentaje) ELSE '1';---------------se
rige por porcentajes.
```

```
END Behavioral;
----------------------------------------------------------------------------
----------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY TRANSMISOR IS

PORT (
            BAUD        : IN STD_LOGIC;
            TxD_INICIO  : IN STD_LOGIC;
            TxD_DATO    : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
            TxD         : OUT STD_LOGIC;
            TxD_BUSY    : OUT STD_LOGIC
);

END TRANSMISOR;


ARCHITECTURE BEHAVIORAL OF TRANSMISOR IS

SIGNAL TxD_AUX_DATO : STD_LOGIC_VECTOR (7 DOWNTO 0);
SIGNAL STATE : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
SIGNAL NEXT_STATE : STD_LOGIC_VECTOR (3 DOWNTO 0):="0000";
SIGNAL TxD_READY : STD_LOGIC;


BEGIN

PROCESS (STATE)
BEGIN
CASE STATE IS

                WHEN "0000" => TxD_READY <= '1';
                WHEN OTHERS => TxD_READY <= '0';

END CASE;

END PROCESS;

PROCESS (STATE)
BEGIN
CASE STATE IS

                WHEN "0000" => TxD_BUSY <= '0';
                WHEN OTHERS => TxD_BUSY <= '1';

END CASE;

END PROCESS;



PROCESS (TxD_READY)
```

```
BEGIN

IF TxD_READY'EVENT AND TxD_READY ='1' THEN

TxD_AUX_DATO <= TxD_DATO;

END IF;

END PROCESS;


PROCESS (BAUD)
BEGIN
IF BAUD'EVENT AND BAUD = '1' THEN

      IF TxD_INICIO = '0' THEN
                  STATE <= "0000";
      ELSE
                  STATE<= NEXT_STATE;
      END IF;
END IF;

END PROCESS;

PROCESS (STATE)
BEGIN
CASE STATE IS

      WHEN "0000" => NEXT_STATE <= "0001";
      WHEN "0001" => NEXT_STATE <= "0010";
      WHEN "0010" => NEXT_STATE <= "0011";
      WHEN "0011" => NEXT_STATE <= "0100";
      WHEN "0100" => NEXT_STATE <= "0101";
      WHEN "0101" => NEXT_STATE <= "0110";
      WHEN "0110" => NEXT_STATE <= "0111";
      WHEN "0111" => NEXT_STATE <= "1000";
      WHEN "1000" => NEXT_STATE <= "1001";
      WHEN "1001" => NEXT_STATE <= "1010";
      WHEN "1010" => NEXT_STATE <= "1011";
      WHEN "1011" => NEXT_STATE <= "1011";
      WHEN OTHERS => NEXT_STATE <= "1011";

END CASE;

END PROCESS;


PROCESS (STATE,TXD_AUX_DATO)
BEGIN
CASE STATE IS

      WHEN "0000" => TxD  <= '1' ;
      WHEN "0001" => TxD  <= '0' ;
      WHEN "0010" => TxD  <= TxD_AUX_DATO(0);
      WHEN "0011" => TxD  <= TxD_AUX_DATO(1);
      WHEN "0100" => TxD  <= TxD_AUX_DATO(2);
      WHEN "0101" => TxD  <= TxD_AUX_DATO(3);
```

```
        WHEN "0110" => TxD  <= TxD_AUX_DATO(4);
        WHEN "0111" => TxD  <= TxD_AUX_DATO(5);
        WHEN "1000" => TxD  <= TxD_AUX_DATO(6);
        WHEN "1001" => TxD  <= TxD_AUX_DATO(7);
        WHEN "1010" => TxD  <= '1';
        WHEN "1011" => TxD  <= '1';
        WHEN OTHERS => TxD  <= '1';

END CASE;

END PROCESS;

END BEHAVIORAL;

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY GEN_FREC IS
PORT
(
            CLK_50      : IN STD_LOGIC;
            F_BASE      : IN INTEGER;
            FREC_OUT    : INOUT STD_LOGIC

);
END GEN_FREC;


ARCHITECTURE BEHAVIORAL OF GEN_FREC IS

SIGNAL COUNT1   : INTEGER := 0;
SIGNAL CLK2     : STD_LOGIC:='0';

BEGIN

PROCESS (CLK_50)
BEGIN

            IF CLK_50'EVENT AND CLK_50 ='1' THEN

                        IF COUNT1 >= F_BASE THEN
                              COUNT1<=0;
                              CLK2<=NOT CLK2;
                        ELSE
                              COUNT1<=COUNT1+1;
                        END IF;
            END IF;
END PROCESS;

FREC_OUT<=CLK2;

END BEHAVIORAL;
```