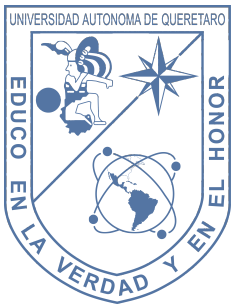


Universidad Autónoma de Querétaro

Facultad de Ingeniería

"Análisis y desarrollo de un controlador de movimiento para un robot PUMA basado en FPGA"

TESIS

Que como parte para obtener los requisitos de

Ingeniero en Automatización

(Sistemas Mecatrónicos)

Presenta

Ramón García Cortés

Centro Universitario

Santiago de Querétaro, Qro.

20 de junio de 2014

México

Universidad Autónoma de Querétaro
Facultad de Ingeniería

Análisis y desarrollo de un controlador de movimiento para un robot PUMA basado en FPGA

TESIS

Que como parte de los requisitos para obtener el grado de

Ingeniero en automatización con especialidad en Mecatrónica

Presenta:
Ramón García Cortés

Dirigido por:
Dr. Juvenal Rodríguez Resendiz

Dr. Juvenal Rodrígez Reséndiz Asesor	_____ Firma
Dr. Edgar Alejandro Rivas Araiza Sinodal	_____ Firma
Dr. Augusto Gómez Loenzo Sinodal	_____ Firma
M. en C. Alfonso Noriega Ponce Sinodal	_____ Firma

_____
Dr. Aurelio González Domíngez
Director de la Facultad

Centro Universitario
Querétaro, Qro.
Julio 2014

Resumen

En este trabajo se presenta un controlador de posición basado en control clásico. Gracias a la flexibilidad del controlador PID este es ampliamente utilizado y aceptado en la industria. Dicho controlador se diseña de manera particular para cada motor mediante la extracción de parámetros de estos realizando un experimento, esta experimentación permite conocer los parámetros de un motor y con los datos obtenidos es posible conocer la planta que representa el motor, con lo cual se puede diseñar un controlador para que cumpla con el requerimiento deseado.

El controlador de movimiento que se propones es un PD para mantener al estabilidad de la planta considerando que es de segundo orden y que siempre será estable.

Se diseño una herramienta de adquisición de datos mediante el FPGA, por medio de esta plataforma es posible realizar el monitoreo casi exacto de la posición del motor en cada instante. Dicha información se procesara en una computadora, que es donde está alojada la técnica de control.

El controlador esta basado en el control clásico, en otras palabras, en el tiempo continuo, por lo tanto se hace uso de métodos numéricos para realizar ciertas operaciones complejas que de otro modo serían tediosas y difíciles de realizar en un FPGA.

A mi madre Estela, todo tu cariño y tu apoyo siempre abundante aún cobija mis sueños, mis metas y mis ganas de salir adelante, nunca estuviste ausente desde que escribí por primera vez con un lápiz. Me enseñaste a creer en mí y a ver la vida a traves de tus manos trabajadoras.

A mi Padre Ramón, mi maestro de la vida, nunca he dejado de aprender de ti. Quien me enseño el valor de trabajar más duro, ser constante y aprendí que el esfuerzo al final se recompensa, has sido el herrero que forjó con el mejor temple mi carácter y mi corazón.

A mi hermana Rosa María, como de costumbre yo siempre pisando las huellas que dejabas para mi, a veces dejadas sin querer, otras veces queriendo, pero que al final solo ambos sabemos sí fueron o no prestadas... más bien, cuáles. Me has enseñado mucho y no dejo de asombrarme cuanto, lo mejor es que sé que contaré contigo siempre hermanita.

A mi tío Isaac, siempre ahí cerca de mi ayudándome, dándome consejos y acompañándome desde que tuve vida, nunca olvidaré lo que me ha enseñado, a ser el mejor y siempre esforzarme.

A mi niña hermosa Marce que amo mucho, mi novia que cuando la conocí me regaló el mejor de los cariños, el más bonito. Tu apoyo incondicional me motiva a ser mejor, a cumplir mis metas y otras metas nuevas. Te admiro mucho y cada día más, la mejor persona que he conocido.

Este trabajo es el sello que da el final de un camino que marca un logro en mi vida, la realización de este proyecto por lo que agradezco a todas aquellas personas que de alguna manera influyeron en la culminación del mismo.

A mis maestros y asesores por todos sus consejos y ayuda que me brindaron para poder realizar este trabajo.

A todos mis amigos y colegas que conocí en el transcurso de mi carrera, por su amistad y apoyo que me brindaron.

Índice general

Indice de tablas

Índice de figuras

Capítulo 1

Introducción

En la industria moderna se vive un fuerte crecimiento en cuanto a la aplicación de tecnología se refiere, pues si se desea ser competitivo entonces se deberá de contar con la tecnología necesaria para lograr la mayoría de sus procesos de manera automática, o bien con un buen plan de manufactura, pues de otro modo no podrá ser altamente competitivo reduciendo sus posibilidades de crecimiento. Lamentablemente la herramienta que que es mas efectiva para mejorar casi cualquier proceso suele ser costosa, por lo tanto para un pequeño empresario; lograr éste cometido se convierte en una labor algo complicada.

Hoy en día la manufactura en México es de muy buena calidad pero esto es en gran parte a las fuertes inversiones que se hacen para impulsar esta mediante la utilización de material y equipo que proviene, en parte, de otras regiones lo cual incrementa el valor sí lo que se desea es invertir en equipo necesario para la industria. Las herramientas más destacadas son las populares tarjetas de control con las cuales se manipulan las máquinas automáticas o en otros casos los equipos robots donde estas tarjetas ya están incorporadas a los robots, en las cuales destacan muchas firmas (Galil MC, Yaskawa, Fanuc, Kuka, por mencionar algunos).

En la actualidad la industria metal mecánica en Querétaro es la base de la economía de este estado obteniendo Producto Interno Bruto (PIB) 19,186,187 miles de pesos solo en el 2009 (INEGI, 2011), lo cual representa el 4.19 % de PIB nacional en la manufactura del

subsector 333-336 de esta clase en el país. Lo cual quiere decir que las máquinas o tarjetas anteriormente mencionadas son muy populares en la industria Queretana, lo cual debido a su costo elevado también el costo final del producto así como el costo de la actualización y/o mantenimiento de dichas líneas de manufactura.

Los servo mecanismos desempeñan un papel importante no solo en la industria, se pueden encontrar en otros lugares, desde un celular hasta algún equipo avanzado de manufactura, sus aplicaciones se han extendido y ahora se consolidan como eje principal de la manufactura con aplicaciones típicas tales como; máquinas automáticas, centros de maquinado, robots, automóviles, etc.

Actualmente existe una gran variedad de herramientas para el control de servo motores los cuales suelen ser con base en microprocesadores (μp), DSP (procesadores de señales digitales), PLC (controladores lógicos programables), FPGA (field programable gate array) por mencionar algunos.

El control de dichos motores suele no ser una tarea sencilla pues cabe mencionar que no todos los motores actúan de la misma manera pues no son de la misma naturaleza, es decir, algunos pueden tener imanes en la armadura para producir un campo, mientras que hay otros cuya armadura debe ser excitada.

1.1. Justificación

Los beneficios de usar tecnología de una velocidad equiparable a un DSP de alto rendimiento pero con un costo mucho menor que además tiene ventajas cuando se trata de realizar tareas concurrentes son cada vez mas evidentes, también las combinaciones de estas, como por ejemplo un FPGA y una computadora donde el FPGA se encarga de la gestión de hardware a la que tenga acceso mientras que la computadora puede realizar tareas de calculo, esto facilita las cosas para un diseñador pues se puede implementar un controlador en un lenguaje de alto nivel como C e implementar este mediante una FPGA.

Ambas tecnologías tienen ventajas y desventajas, en este caso el desarrollo de

un controlador de posición para un motor de DC puede aportar a nuevas generaciones información, metodología, referencias para algún desarrollo futuro.

También es importante mencionar las múltiples aplicaciones didácticas que puede ofrecer esta herramienta gracias a que está basada en el dispositivo FPGA, un elemento utilizado ampliamente por los estudiantes durante su formación académica, este dispositivo puede utilizarse como una base, de tal manera que sobre este se puedan añadir las nuevas ideas sin que estas se vean retrasadas por replantar un problema que aquí se busca resolver.

La idea entonces es la realización de un control de servosistemas embebido en C y que en combinación con un FPGA se le adecue las capacidades mínimas requeridas para dicho fin, además de que sea facilmente desarrollable se espera que esta modalidad alcance un costo razonable y con una eficiencia adecuada para casi cualquier servomecanismo.

1.2. Planteamiento del problema

Quizás la descripción de hardware es una labor mas tediosa que programar un MCU (Micro Controller Unit), aunque en algunas ocasiones quizás puede llegar a ser mas simple debido al tipo de tareas que se demandan.

La descripción de hardware ofrece una independencia en cuento al diseño se refiere, es decir que se puede adecuar a algún estilo de desempeño como uno donde el control sea independiente para cada eje o motor (Masatoshi Nakamura, 2004), y es por eso que se ha optado por el uso de la herramienta FPGA con la cual es posible controlar todos los motores disponibles al mismo tiempo.

Una ventaja importante que ofrece esta plataforma es la flexibilidad que posee debido a su arquitectura, eso permite realizar una actualización y/o modificación de la estructura del control puesto que un FPGA se puede reconfigurar sin necesidad de construir un nuevo prototipo, esto permite al usuario final mantener su sistema siempre en óptimas condiciones cumpliendo los requerimientos que cada vez son más rigurosos, De esta manera se logra evitar la constante actualización de hardwere que exige la cada vez más demandante

industria, reflejándose en un ahorro para el usuario final.

Es importante mencionar que el código VHDL es portable, en otras palabras, si una nueva tecnología emerge al mercado y esta llegara a remplazar la actual; no abria problemas de migración de codigo, pues este es totalmente portable entre FPGAs sin importar quien sea fabricante.

1.3. Hipótesis y objetivos

La descripción de hardware es una linea de investigación y aplicación de la electrónica digital, su campo de aplicación tiene ventajas en cuanto a funciones concurrentes se trata, pues se pueden hacer múltiples tareas a la vez como revisar la posición de un encoder fácilmente mientras se realiza otro proceso como una transferencia de datos, es decir se podrá hacer la parte crítica en tiempo real con los componentes mínimos.

Algunas ventaja de un lenguaje de programación como C es que se pueden implementar algoritmos de una manera más rápida e intuitiva, aunque no tenga un rendimiento excepcional es suficiente como para calcular un controlador. Esto permitiría modificar algún algoritmo en menos tiempo que una descripción de hardware mediante VHDL.

1.3.1. Hipótesis

Se puede desarrollar un sistema con una computadora y un FPGA combinada capaz de realizar la tarea de controlar la posición de un servo motor de CD de imán permanente.

1.3.1.1. Objetivo general

Desarrollar una herramienta de control con base en un sistema combinado de una computadora y un FPGA para el control de movimiento de un servo motor que permita realizar acciones de control controlar un motor y que este sea confiable además de económi-

co.

1.3.1.2. Objetivos particulares:

- Desarrollar un prototipo electrónico con base en FPGA para controlar el voltaje en la armadura del motor.
- Desarrollar una etapa de potencia simple para manipular el voltaje de alimentación del motor.
- Implementar un protocolo de comunicación RS-232 entre el controlador o computadora y el FPGA.
- Diseñar un controlador adecuado para un servo motor.
- Implementar el control PD en el sistema.

Capítulo 2

Revisión de literatura

2.1. Antecedentes

Tanto el servo sistema mecatrónico como el sistema de control satisfacían las condiciones de movimiento de los ejes de transferencia de las máquinas de control numérico fue originalmente (al rededor de 1967) creado cuando se desarrollaba el servo motor de corriente directa. Después en 1975 por *YASKAWA Electric*, fue vendido el sistema de control de velocidad (servo driver unit) unificado con el sistema de compensador de control y los amplificadores de potencia (Masatoshi Nakamura, 2004).

Al comienzo fue principalmente adoptado por el eje de control de transferencia de una maquina trabajando. Desde 1980 fue también adoptada por los controles de velocidad y posición de varias firmas de estos mecanismos como las de robots industriales. En la generación de este Servo sistema mecatrónico, el patrón de control como punto de inicio de la construcción del Servo Sistema, está de acuerdo como sigue.

1. La perturbación en la referencia de torque es incrementada en velocidades por debajo de 1 RPM.

2. La relación del control de velocidad es de 1 hasta varios miles (mínimo 1 RPM y máximo $3000 - 5000$ RPM).

3. La relación aceleración/desaceleración tiene un comportamiento efectivo en valores por debajo del umbral de corriente definido por el usuario.

Con un bajo costo y habilitado con una alta velocidad de adquisición de datos y hardware para el procesamiento, el control digital se ha convertido en una herramienta importante para el análisis, diseño e implementación en sistemas de control. Un típico sistema de control se muestra en la Figura 2.1 donde las señales en el tiempo continuo son filtradas por un filtro pasa bajas y muestreadas por un convertidor analógico-digital (ADC) a señales de tiempo discreto. Estas señales de tiempo discreto son después procesadas por un microprocesador para así generar la salida del control en el tiempo discreto. El convertidor digital analógico (DAC) se encarga de convertir la señal de tiempo discreto en otra pero de tiempo continuo. En ocasiones se agregan filtros para suavizar la señal con el fin de eliminar armónicos presentes en el filtro (Bogdan M. Wiliamowski, 2011).

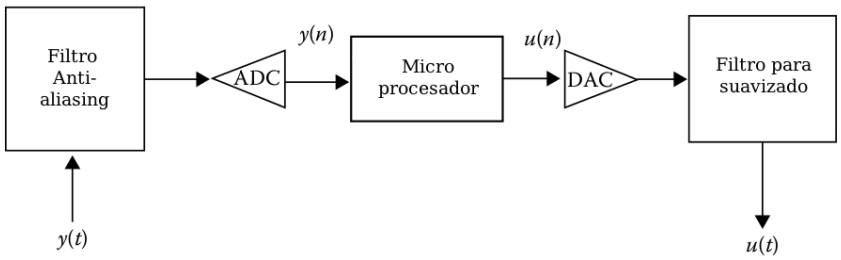


Figura 2.1: *Sistema digital de procesamiento*

Otro trabajo enfocado al control de servosistemas como lo es el de Colín Robles (Ángel Colín Robles, 2006), en el cual describe los módulos requeridos para desarrollar un control PID además de proponer un medio de comunicación entre este y una computadora. En el trabajo de (savita Sonoli, 2010) se enfoca únicamente a controlar la velocidad de un motor en función de la corriente que pasa por el, de modo que si se desea aumentar la velocidad de dicho motor se procede a incrementar el voltaje hasta alcanzar un valor de corriente previamente calculado y necesario para alcanzar la nueva velocidad, en este trabajo se hace uso de un FPGA para la implementación del controlador pues de acuerdo con el autor esto significa una ventaja económica debido a que todo el controlador está embebido, también usa un Convertidor Analógico Digital (ADC por sus siglas en ingles) con el que

mide la variable a controlar que en este caso es la corriente, la cual logra medir mediante un arreglo de resistencias.

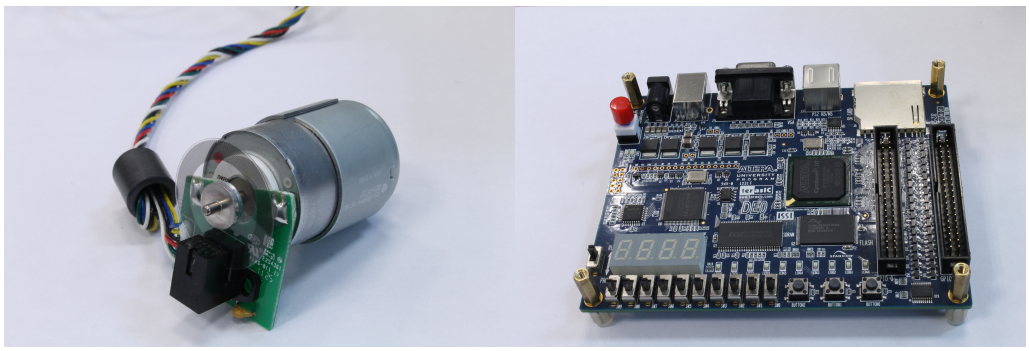
En los trabajos (Wahmad, 2008),(Osman Koct, 2011),(Luis F. Castaño Londoño, 2011) se hace uso de un ancho de pulso modulado (PWM, por sus siglas en ingles) el cual es recibido o por un circuito que convierte esta señal en una señal analógica mediante electrónica o bien es recibida por un Servo Driver que admite PWM como referencia. También cabe mencionar que en el primero de estos trabajos se hace uso de un microcontrolador con el cual se encarga de controlar únicamente la velocidad de un motor de CD, mientras que en estos dos últimos trabajos se controla la posición además de la velocidad, también en el trabajo (M.Sivaramakrishna, 2011) se controla la posición y velocidad de un servomotor donde además se realizó un análisis para el ajuste de los polos del sistema.

Capítulo 3

Metodología

En la experimentación se hizo uso del conocimiento que ya se venia estudiando previamente en la licenciatura además de que se puntualizarán algunos detalles en cuanto a lo que se realizó y la forma en que se hizo.

La experimentación se realizó en la facultad de ingeniería de la Universidad Autónoma de Querétaro en el laboratorio de Mecatrónica, se empleo un motor de 12 v CD con imán permanente en la armadura:



(a) Servo motor (b) FPGA

Figura 3.1: *Material utilizado*

3.1. Marco teórico

El concepto fundamental del control de movimiento de los servomotores no ha cambiado en muchos años. La razón básica para uso de servosistemas en contraste con los sistemas de lazo abierto incluyendo la necesidad de mejorar el tiempo de respuesta transitoria, errores en el estado estacionario y reducir la sensibilidad a perturbaciones.

Incrementar la respuesta transitoria significa incrementar el ancho de banda del sistema. una rápida respuesta significa una rápida estabilización permitiendo un alto rendimiento en el motor. Reducir el error en el estado estacionario refleja exactitud en el servosistema. Finalmente reducir la sensibilidad a las perturbaciones significa que el sistema puede tolerar fluctuaciones tanto en la entrada como la salida de parámetros. Un parámetro de entrada puede ser un valor de voltaje. Un ejemplo de una fluctuación en los parámetros es el cambio en tiempo real de inercia de carga o masa y perturbaciones inesperadas en el torque de la flecha Kariyappa B. S. (2009).

3.1.1. Sistemas de 2do. orden

En la Figura 3.2 se muestra un sistema de segundo orden con un controlador proporcional en cascada y una realimentación unitaria.

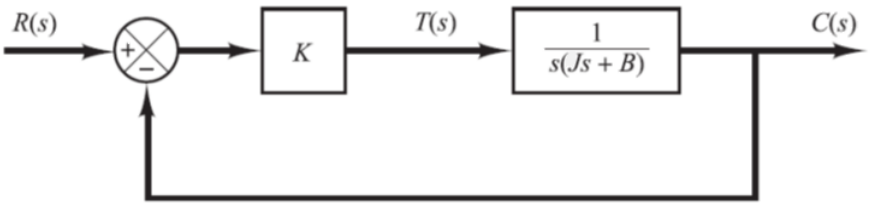


Figura 3.2: *Sistema de 2do. orden con un controlador proporcional.*

Si se aplica las reglas de reducción de bloques; la de $G(s)H(s)$ por estar en cascada y la de $\frac{G(h)}{1+G(s)H(s)}$ respectivamente, entonces tendremos que la función de transferencia es:

$$\frac{C(s)}{R(s)} = \frac{K}{Js^2 + Bs + k} \tag{3.1}$$

Otra forma de ver la función de transferencia es completando el binomio cuadrado perfecto del denominador:

$$\frac{C(s)}{R(s)} = \frac{\frac{K}{J}}{\left[s + \frac{B}{2J} + \sqrt{\left(\frac{B}{2J}\right)^2 - \frac{K}{J}}\right]\left[s + \frac{B}{2J} - \sqrt{\left(\frac{B}{2J}\right)^2 - \frac{K}{J}}\right]} \tag{3.2}$$

De esta manera se puede observar la condición de los polos, es decir, sí estos son complejos si $B^2 - 4JK < 0$ y serán reales sí $B^2 - 4JK >= 0$. Esto es importante para determinar el tipo de respuesta transitoria que se puede obtener de este sistema pues hay una serie de reglas en las que se puede apoyar para determinar dicha respuesta. (Ogata, 2010).

La forma estándar de un sistema de segundo orden esta descrita por la Ecuación 3.3:

$$\frac{C(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{3.3}$$

A partir de la Ecuación 3.3 y la Ecuación 3.1 se puede determinar que $\frac{K}{J} = \omega_n^2$, $\frac{B}{J} = 2\zeta\omega_n = 2\sigma$, donde ζ es el factor de amortiguamiento y ω_n es la frecuencia natural no amortiguada y 2σ también se le conoce como atenuación.

En los sistemas de segundo orden hay casos de respuesta de acuerdo con el valor de ζ con el que se cuente (Ogata, 2010).

- Caso no amortiguado $\zeta = 0$, no hay amortiguamiento por lo tanto la respuesta del sistema es oscilatoria y no se amortigua, aquí los polos son puramente imaginarios y conjugados.

- Caso sub amortiguado $0 < \zeta < 1$, la respuesta es oscilatoria y además amortiguada, los polos son complejos conjugados.

- Caso críticamente amortiguado $\zeta = 1$, la respuesta no oscila y es lenta, los polos son reales e iguales.

- Caso sobre amortiguado $\zeta > 1$, la respuesta no oscila y es muy lenta, los polos son reales y diferentes.

La función de transferencia de un sistema de 2do. orden, aplicando teoría de fracciones parciales y para una entrada tipo escalón, quedaría de la siguiente manera:

$$C(s) = \frac{1}{s} - \frac{s + \zeta\omega_n}{s^2 + 2\zeta\omega_n + \omega_n^2} - \frac{\zeta\omega_n}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{3.4}$$

Recordar que:

$$\omega_d = \omega_n\sqrt{1 - \zeta^2}$$

Aplicando la transformada inversa de Laplace y reduciendo obtenemos la Ecuación 3.5:

$$c(t) = 1 - \frac{d^{-\zeta\omega_n t}}{\sqrt{1 - \zeta^2}} \sin\left(\omega_d t + \tan^{-1}\frac{\sqrt{1 - \zeta^2}}{\zeta}\right), para t \geq 0 \tag{3.5}$$

La respuesta transitoria esta descrita por la Figura 3.3.

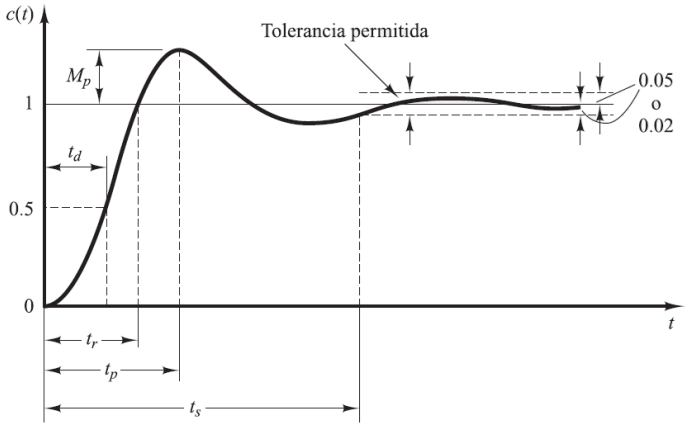


Figura 3.3: *Curva de respuesta a un escalón unitario (Ogata, 2010).*

3.1.2. Controlador PID

Los sistemas de control proporcional integral y derivativo (PID) son suficientes para resolver el problema de control de muchas aplicaciones en la industria, particularmente cuando la dinámica del proceso lo permite (en general procesos que pueden ser descritos por sistemas de primer y segundo orden, los fabricantes proporcionan los controladores PID de variadas formas). Existen sistemas del tipo independientes "standalone", con capacidad de controlar uno o varios lazos de control, aunque estos también pueden estar embebidos como parte del equipo.

Una de las principales razones que el controlador PID continúe siendo ampliamente utilizado se dio cuando la industria se trasladó del control analógico al control digital, nuevas características como el ajuste de las ganancias a través de un puerto de comunicación (RS-232/485, USB, Ethernet, CAN, PCI entre otros) tuvieron un gran impacto en las tecnologías y los métodos de control industrial, dando paso cada día a sistemas de control más sofisticados que los controladores estudiados en la teoría de control clásico, estos son los sistemas de control inteligente, controles de lógica difusa (Fuzzy), redes neuronales y algoritmos genéticos, a pesar de los avances en el área de control el controlador clásico PID. La Ecuación 3.6 muestra una ecuación PID en el tiempo continuo. (Ogata, 2010).

$$V_a(t) = R_a I_a(t) + L_a \frac{\mathrm{d}I_a(t)}{\mathrm{d}t} + K_b \omega(t) \tag{3.6}$$

Donde K es la ganancia, T_i es el tiempo de integración, T_d es el tiempo de derivación o tiempo de carga.

El controlador PID sigue siendo ampliamente utilizado, dadas las siguientes ventajas: por su simplicidad, ya que no requiere del conocimiento del modelo matemático de la planta, ningún diseño a analizar. El usuario con un simple ajuste en tres ganancias puede lograr la respuesta deseada del sistema, además existen un gran número de herramientas de software para diseñar el controlador, evaluar la estabilidad y rendimiento, así como poder realizar una simulación. Aunque la mayoría de los controladores industriales hoy en día

son controladores del tipo PID o variantes del mismo, existen limitaciones estos tienen una dinámica de 2do. orden, por lo que no son adecuados para compensar plantas de un orden mayor puesto a que existen un grupo de plantas inestables que no pueden ser estabilizadas por ningún miembro de la familia PID.

La industria ha desarrollado variantes e interesantes mejoras en el controlador PID para movimiento de motores. Una de las piezas del controlador que ha mostrado cambios y mejoras es el sistema de retroalimentación. En el Tabla 3.1 se muestras los dispositivos de retroalimentación comúnmente disponibles para el control de motores.

Variable retroalimentada	**Fuente**
Fuerza Electromotriz	Medición de voltaje, calculada a partir de una señal PWM o la velocidad
Corriente	Sensor de efecto Hall, resitencia Shunt o toroide
Aceleración	Encoder o Resolver, odómetro
Velocidad	Encoder, Resolver o tacómetro
Posición	Encoder, Resolver, potenciómetro o un trasformador lineal diferencial variable (LVDT)

Tabla 3.1: *Variables usualmente retroalimentadas*

La velocidad de un motor de CD es controladoa mediante la alimentación de voltaje a la armadura del motor. El voltaje de corriente directa produce una corriente I_a variable en la armadura, que a su vez crea un campo magnético entre el rotor y estator. Cabe mencionar que la corriente requerida por la armadura no puede ecceder la máxima capacidad de corriente que provee el amplificador de potencia. El limite de corriente es tratada como no linealidad en un sistema de control de un motor de CD. Es evidente que habrá un pico ce corriente por un corto periodo de tiempo durante la aceleración y frenado del motor. El pico de corriente y su duración son dados por los fabricantes de los amplificadores.El flujo magnetico es constante mientras que el voltaje en la armadura es variable en el control por armadura en los motores de CD. El campo magnético produce un torque, el cual es usado para rotar el rotor que a su vez está conectado a la flecha del motor. En los motores de CD, una fuerza contra electro motriz V_b, la cual es proporcional a la velocidad del rotor, es producida en el circuito del motor. Altintas (2000)

3.1.3. Servo mecanismo

Un servomecanismo es un sistema formado de partes mecánicas y electrónicas que en ocasiones son usadas en robots, con parte móvil o fija. Puede estar formado también de partes neumáticas, hidráulicas y controlado con precisión. Ejemplos: brazo robot, mecanismo de frenos automotor, etc.

Figura 3.4: *Servomecanismo*

3.1.4. Diseño del controlador

Un controlador puede ser representado como en la Ecuación 3.7.

$$u(t) = K_p e(t) + K_i \int e(t) + K_d \frac{de(t)}{dt} \tag{3.7}$$

Donde K_p, K_i, K_d son los coeficientes del controlador y $e(t)$ es el error y $u(t)$ la entrada de la planta. Estos coeficientes deben ser seleccionados adecuadamente para obtener una respuesta favorable en la planta de acuerdo con las características requeridas en un dado caso.

Puede observarse que la estructura de estos filtros está basada en operaciones de adición sustracción y multiplicación, procesos que pueden implementarse digitalmente

mediante métodos numéricos se pueden trasladar operaciones como la derivada y la integral a un procesador digital, que puede ser una computadora, un MCU, un FPGA, etc.

Tradicionalmente existen dos alternativas para la implementación digital de controladores: la ejecución en procesadores de propósito general o la realización en hardware a la medida. La primera es económica y flexible, pero conlleva velocidades de procesamiento lentas, aunque tiene la ventaja de que se puede programar de una manera mas sencilla. La segunda generalmente se usa cuando se requiere un alto desempeño lo que implica un aumento considerable en el costo reservándose para aplicaciones de uso masivo o aplicaciones particulares y suelen ser poco flexibles. La lógica programable, con herramientas como los lenguajes de descripción de hardware y los potentes dispositivos con capacidades cada vez mayores, como los son las FPGA, permite combinar velocidades altas de procesamiento y flexibilidad de hardware, por lo tanto constituye una opción viable para el desarrollo de controladores digitales o complementos de estos controladores.

3.1.5. Servo sistema

El servosistema (o servomecanismo) es un sistema de control retroalimentado en el que la salida es algún elemento mecánico, sea posición, velocidad o aceleración como se muestra en la Figura 3.5.

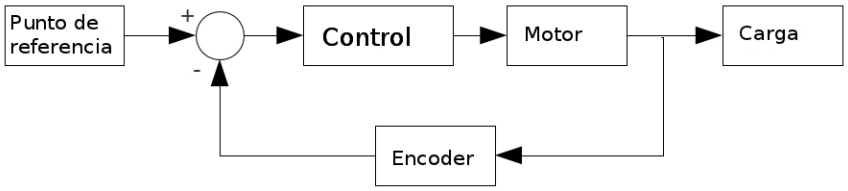


Figura 3.5: *Lazo de control de un servosistema*

En la industria un servocontrol puede llegar a ser tan complejo como sea necesario para cumplir con las expectativas de rendimiento, pudiendo tener no solo un controlador para el voltaje de la armadura, sino uno para la corriente enviada al motor, velocidad, entre otras cosas como se muestra en la Figura 3.6, Masatoshi Nakamura (2004).

En la Figura 3.7 se puede observar la estructura completa de un servo sistema

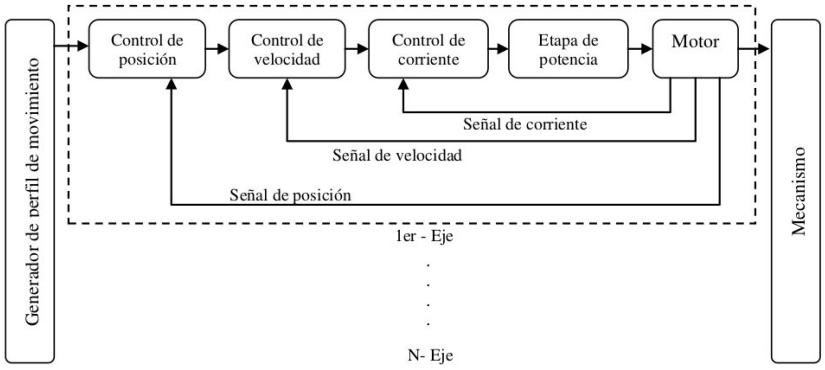


Figura 3.6: *Estructura de un servosistema industrial*

mecatrónico donde este sistema incluye la parte del mecanismo, el eje del servo motor incluido en la parte mecánica, la parte que controla el sistema está separada del generador de la referencia de entrada. El servo sistema en cada eje está constituido por el motor, la etapa de potencia "amplificadores", la etapa de control de corriente y la parte de control de posición y el sensor (de posición, de velocidad y de corriente) con la finalidad de detectar la señal desde varias partes y conectada con la parte del mecanismo por el hardwere Masatoshi Nakamura (2004).

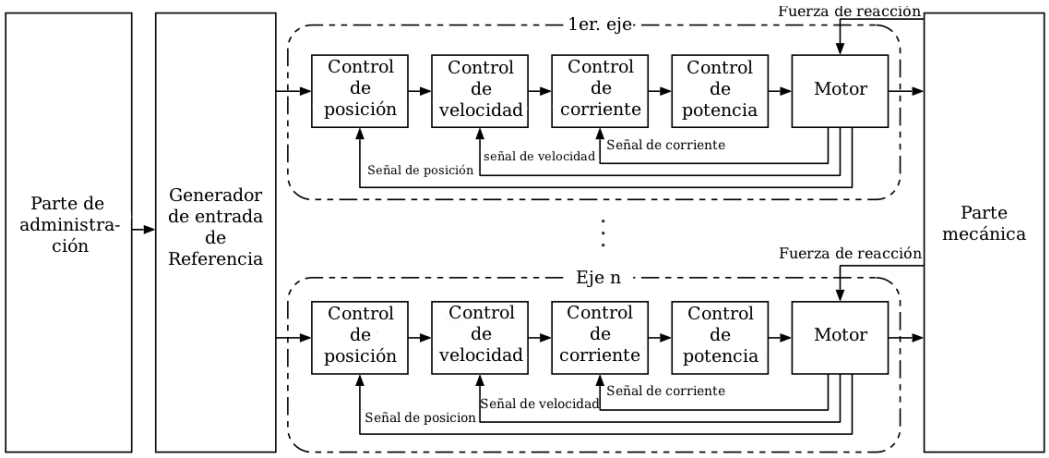


Figura 3.7: *Estructura de un servo sistema Mecatrónico industrial*

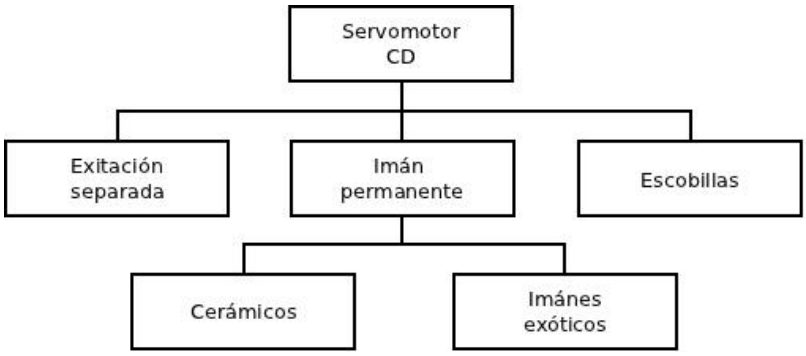


Figura 3.8: *Tipos de servomotores y caracteristicas*

3.1.6. Servo motores

Los servo motores fueron diseñados para generar un torque elevado al inicio del movimiento lo que les permite ubicarse con precisión de acuerdo a los requerimientos del usuario, esto es debido a la naturaleza de su estator y rotor. Es por ello que son ampliamente utilizados en muchos dispositivos de uso común desde una simple y pequeña impresora hasta autos o grandes máquinas industriales tales como los centros de maquinado y robots. La respuesta tan uniforme de estos elementos es gracias a las características de su estator, esto es gracias a el numero de bobinas que hay en el estator ya que estas permiten que se logre un par de salida mas suave. Conforme acelera una fuerza contra electro motriz se genera la cual reduce la corriente. la operación de todos los servo motores DC es similar. Firoozian (2009)

En la Figura 3.8 se muestran en resumen los motores de corriente directa disponibles en el mercado y en el Tabla 3.2 se muestran los tipos de motores y sus características Firoozian (2009).

3.1.7. Servo amplificador

Además del motor, es necesario un sistema amplificador que convierta la señal de posición en una con suficiente potencia para mover al (servo) motor. Por ejemplo, en el caso de un motor eléctrico, el amplificador deberá proveer suficiente corriente (continua o alterna, según se trate) para manejar o controlar gradualmente la velocidad, hasta llegar

Tipo de motor	*Fuente de alimentación*	*eficiencia típica*	*Acoplamiento*	*Controlador*
DC con escobillas	DC	$< 50\,\%$	Directo o con reductor	Simple o complejo
DC sin escobillas (BLCD)	Frecuencia variable, 3 faces AC	$> 90\,\%$	Directo o con reductor	Complejo
AC inducción	3 fases AC	$< 90\,\%$	Reductor	Simple
AC síncrono	Frecuencia variable, 3 fases AC	$> 90\,\%$	Directo o con reductor	Simple o complejo
Motor a pasos	DC conmutación digital	$< 5\,\%$	Directo o con reductor	Simple

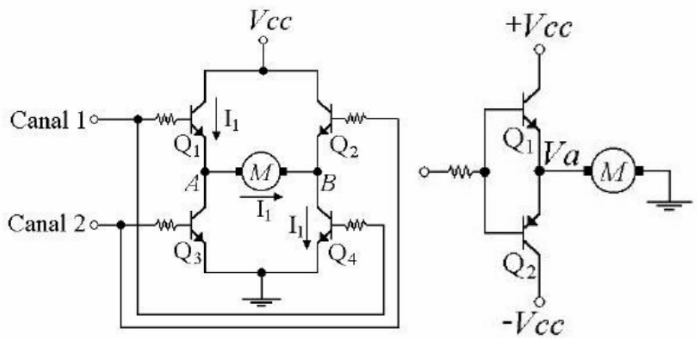
Tabla 3.2: *Tipos comunes de motores y sus características*

a los parámetros fijados por dicho controlador.

3.1.7.1. Servo amplificador lineal

Se pueden encontrar dos tipos de servo amplificadores lineales: el H y el T; estos se muestran en las Figura 3.9a y 3.9b , respectivamente, y tienen su base en transistores de unión bipolar, aunque pueden implementarse con otro tipo de dispositivos semiconductores de potencia como los MOSFET (Metal-Oxide-Semiconductor Field-Effect Transistor) el tipo H, en ocasiones llamado amplificador puente, tiene la ventaja de requerir un suministro unipolar de CD. Sin embargo, no siempre es fácil de operar en un modo lineal porque el motor esta "flotado" con respecto a la tierra eléctrica del sistema, y debido a esto la retroalimentación de corriente y voltaje de armadura no son fáciles de lograr. La descripción del funcionamiento del tipo H es la siguiente: un par de transistores de unión bipolar (o dispositivos MOSFET) se turnan para operar en la región activa, esto es, Q_1 y Q_4 o Q_2 y Q_3, haciendo que el voltaje de armadura V_{AB} genere un flujo de corriente de armadura contrario a I_1, el motor girará ahora en sentido contrario a las manecillas del reloj. El valor instantáneo de V_{AB}, y en consecuencia, la velocidad del motor, dependerá de la cantidad de corriente aplicada a la base de los transistores, determinada por el circuito de control que precede a la etapa del amplificador de potencia.

El segundo tipo general de servo amplificador, es el tipo T, que requiere de una fuente bipolar de DC, es más fácil de controlar, ya que el motor no está flotando con respecto a la tierra eléctrica, y la retroalimentación de corriente y voltaje de armadura es más sencillo de implementar.



(a) Tipo H (b) Tipo T

Figura 3.9: *Servo amplificadores lineales.*

3.1.7.2. Servo amplificador tipo PWM

Una de las mayores desventajas con los amplificadores lineales, se refiere a que la salida es solamente una fracción del voltaje total de suministro, por ejemplo, durante las partes iniciales o finales de un movimiento o cuando éste se realiza deliberadamente a velocidades bajas; lo anterior se debe a la operación de los transistores de potencia, en su región activa (o MOSFETs en su región de saturación), lo que significa que la caída de tensión que se lleva a cabo entre colector y emisor V_{CE} del transistor es considerable.

Por consiguiente, la potencia disipada en el colector, producto de corriente de colector y V_{CE}, puede ser grande del orden de 100 W, así los transistores y disipadores de calor deben seleccionarse adecuadamente. Afortunadamente ahora es posible usar un enfoque diferente que es generalmente más rentable, técnicas como el PWM, nos permiten controlar la potencia entregada a la carga.

Así como con los servoamplificadores lineales, los dispositivos PWM, pueden ser del tipo H o T y presentan las ventajas y desventajas de los mismos. Sin embargo, a

diferencia del caso lineal, el voltaje de salida del circuito T o H puede ser casi igual, al valor total del voltaje de suministro positivo o negativo de la fuente de DC.

3.1.8. Encoder

El encoder es un transductor rotativo que transforma un movimiento angular en una serie de pulsos digitales como se muestra en la Figura 3.10. Estos pulsos digitales pueden ser utilizados para encontrar un desplazamiento angular o lineal si se asocian a cremalleras o a husillos. Las señales eléctricas de rotación pueden ser elaboradas mediante controles numéricos (CNC), controladores lógicos programables (PLC), sistemas de control etc. Las aplicaciones principales de estos transductores están en las máquinas herramienta o de elaboración de materiales, en los robots, en los sistemas de motores, en los aparatos de medición y control Eltra (2000).

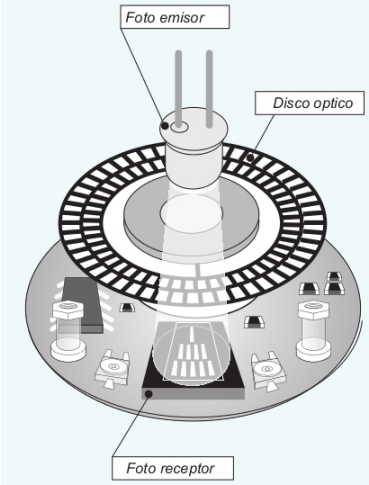


Figura 3.10: *Encoder tipo rotativo, se aprecian los canales A, B y Z*

3.1.9. PWM

También conocido como modulación de ancho de pulso, es la base del control en la electrónica de potencia. El teórico levantamiento en cero y el tiempo de bajada de una forma de onda PWM ideal representa la forma preferida de controlar dispositivos modernos

de semiconductores de potencia. Se busca que dicho PWM sea tan rápido como sea posible para evitar perdidas debido al tiempo de transición o perdidas asociadas a la conmutación.

La frecuencia del pulso es de lo mas importante cuando se define un metodo por PWM y puede ser constante o variable. Un pulso binario PWM se puede describir como en la Ecuación **??** de nanera matemática.

$$b_{pwm}(t) = sgn[r(t) - c(t)] \tag{3.8}$$

Donde sgn es la señal función seno.

Para el PWM constante se pueden montar diferentes señales portadoras, como se muestra en la Figura 3.11 donde se aprecian dientes de cierra y una señal triangular, en otras palabras se puede decir que es con estas señales con las que se compara un valor Duty_cycle para ver si es uno o cero el PWM.

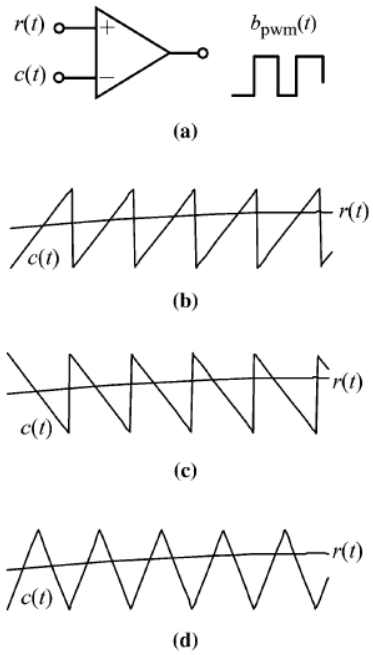


Figura 3.11: *Señales portadora de PWM.*

3.2. Identificación de parámetros del motor

El modelo dinámico de un motor de CD depende de propiedades que se pueden dividir en dos tipos; eléctricas y mecánicas, dicha propiedades son; resistencia hom e inductancia H de la armadura ademas de un voltaje contra electro motriz presente en la parte eléctrica y para la parte mecánica; la inercia de la armadura, fricción mecánica que depende de la velocidad y el par mecánico.

La fricción de un motor puede provenir de diferentes fuentes; viscosidad en rodamientos o en caso de tener un enfriamiento por aire forzado. Estas se consideran como fuerzas mínimas que pueden considerarse despreciables puesto a que pueden llegar a ser muy pequeñas de tal modo que su ausencia en el modelo no implique una perdida de información significativa de este. En la parte electrica también se puede considerar como despreciable la inductancia en la armadura pues su valor es tan pequeño que es poco representativo en el modelo.

Partiendo de lo antes mencionado se considera la Figura 3.12 con las condiciones previamente mencionadas para su evaluación.

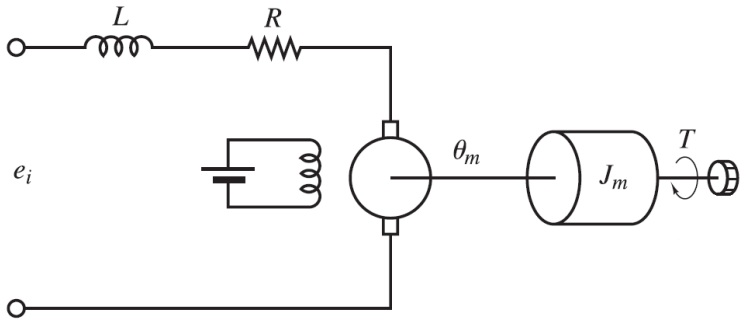


Figura 3.12: *Diagrama fisico de un motor de CD.*

La dinámica eléctrica está dada por las ecuaciones 3.9, 3.10 y 3.11.

$$\frac{\mathrm{d}i(t)}{\mathrm{d}t}L + Ri(t) + e_p(t) = \quad u(t) \tag{3.9}$$

$$k_b\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \quad e_p(t) \tag{3.10}$$

$$i(t)k_T = \quad T \tag{3.11}$$

La dinámica mecánica está dada por las ecuaciones 3.12 y 3.13.

$$J\frac{\mathrm{d}^2\theta(t)}{\mathrm{d}^2t} + b\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \quad T \tag{3.12}$$

$$k_T i(t) = \quad T \tag{3.13}$$

Resolviendo el sistema de ecuaciones para obtener una función de transferencia cuya relación sea $\frac{\theta(s)}{u(s)}$:

Sustituyendo 3.13 en 3.12 se tiene:

$$J\frac{\mathrm{d}^2\theta(t)}{\mathrm{d}^2t} + b\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = k_T i(t)$$

Aplicando La Place se obtiene:

$$s^2\theta J + sb\theta = k_T I$$

$$I = \frac{s^2\theta J + sb\theta}{k_T}$$

Resolviendo el sistema de ecuaciones 3.9- 3.11:

$$\frac{\mathrm{d}i(t)}{\mathrm{d}t}L + Ri(t) + k_b\frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = \quad u(t)$$

Aplicando la transformada de La Place:

$$sIL + RI + K_b s\theta = \quad U$$

Combinando la parte mecánica con la parte eléctrica del motor:

$$\frac{s^2\theta J + sb\theta}{k_T}(sL+R) + k_b s\theta = U$$
$$\theta\left(\frac{s^2 J + sb}{k_T}\right)(sL+R) + k_b s\theta = U$$
$$\frac{\theta}{U} = \frac{1}{\left(\frac{s^2 J+sb}{k_T}\right)(sL+R) + k_b s}$$
$$\frac{\theta}{U} = \frac{k_T}{(s^2 J + sb)(sL+R) + k_b k_T s}$$

$$\frac{\theta}{U} = \frac{k_T}{s\left[(sJ+b)(sL+R) + k_b k_t\right]} \tag{3.14}$$

Debido a que la inductancia es muy pequeña se puede despreciar, esto debido a que su ausencia no afecta en gran medida el comportamiento de la planta. De este modo la Función de Transferencia que dará descrita por las Ecuación 3.15.

Entonces, partiendo de la Ecuación 3.14:

$$\frac{\theta}{U} = \frac{k_T}{s\left[(sJ+b)(R) + k_b k_t\right]}$$
$$\frac{\theta}{U} = \frac{\frac{k_T}{JR}}{s\left[\left(s+\frac{b}{J}\right) + \frac{K_b k_t}{JR}\right]}$$

$$\frac{\theta}{U} = \frac{\frac{k_T}{JR}}{s\left(s + \frac{Rb+K_b k_T}{JR}\right)} \tag{3.15}$$

Para fines prácticos se sustituirán algunas partes de la Ecuación 3.15 con las Ecuaciones 3.16 y 3.17

$$\frac{k_T}{JR} = C_k \tag{3.16}$$

$$\frac{Rb + k_b k_T}{JR} = C_p \tag{3.17}$$

Entonces la Ecuación 3.15 se reduce a 3.18

$$\frac{\theta}{U} = \frac{C_k}{s(s + C_p)} \tag{3.18}$$

Debido a que por su propia naturaleza un motor de CD es un sistema sobreamortiguado, por lo tanto no presenta oscilaciones cuando se le es aplicado una señal de referencia de tipo escalón unitario. Esto resulta poco útil si lo que se desea es determinar los parámetros del motor; en este caso las constantes C_k y C_p, las cuales ya engloban en ellas mismas los valores de la resistencia R, inercia J, constante de torque k_T y la constante de fricción K_b.

En el análisis de sistemas de segundo orden se determinó que el amortiguamiento depende de dos factores, primeramente de la constante factor de amortiguamiento ζ y le secunda en relevancia la frecuencia natural no amortiguada ω_n. A su vez ω_n tiene que ver con la ganancia del sistema. Debido a que no se puede variar el factor de amortiguamiento; este si se puede calcular gracias a que dicho factor se ve reflejado en el tipo de respuesta transitoria, es decir; hay una relación entre el sobrepaso con este factor. Entonces sí es posible conocer dicho factor además de la ganancia que se le aplicaría al sistema para que este no sea sobre amortiguado, sino que en vez de eso sea subamortiguado, entonces es posible también conocer el factor de frecuencia natural no amortiguada ω_n realizando un poco de álgebra.

Entonces para el experimento se propone un sistema como el que se muestra en la Figura 3.13.

Para que se pueda obtener el factor de amortiguamiento del motor es necesario

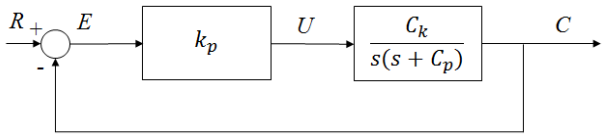


Figura 3.13: *Motor con un controlador tipo P realimentado.*

que la respuesta transitoria presente oscilaciones, entonces; tomando en cuenta que el maximo voltaje que se puede proveer debido a la fuente de alimentación y perdidas en el Puente H es aproximadamente 10V, por tanto se propone usar una ganancia K_p de 3.2 para obtener una señal de control cuando el error sea igual a el valor de referencia de $9 < valordereferencia \times 3.2 < 10V$. No debe de ser mayor a 10V por que sería imposible conocer el sobrepaso real del motor sí es que la señal de control se llegase a saturar.

Aplicando las reglas de reducción de bloques se llega a un sistema como el de la Figura 3.14.

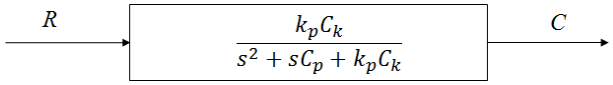


Figura 3.14: *Diagrama de bloques reducido para un motor con un controlador P.*

Entonces, se tiene que:

$$C_p = 2\zeta\omega n \tag{3.19}$$

$$C_k = \frac{\omega n^2}{k_p} \tag{3.20}$$

Entonces se realiza una prueba con la planta del motor con un controlador tipo P con una señal de entrada tipo escalón de 3.14159 y una ganancia k_p de 3.2.

Como resultado se obtiene la gráfica de la Figura 3.15.

A partir de la gráfica se puede determinar dos características de interés; el sobre paso máximo y el tiempo de levantamiento. Estos valores son: $M_P = 5.351$ y $t_r = 0.0288$

Considerando las Ecuaciones 3.21, 3.22, 3.23 y 3.24.

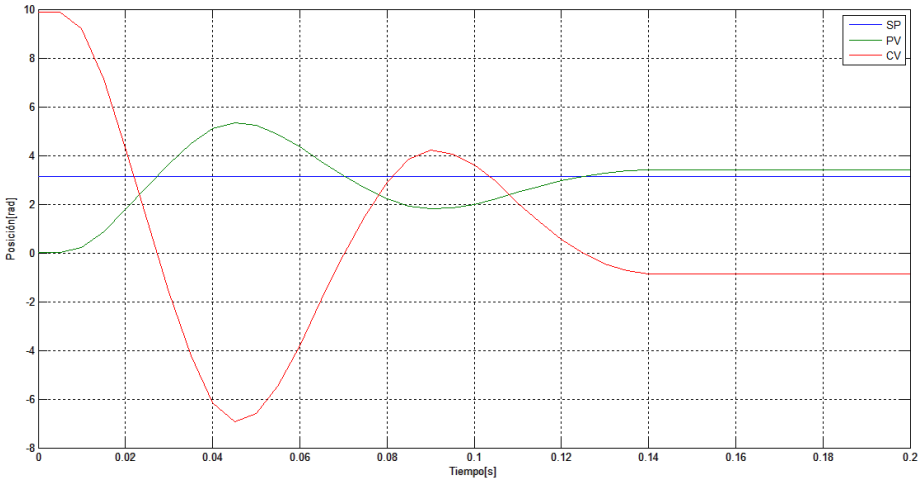


Figura 3.15: *Respuesta transitoria de motor a un impulso escalón de 3.1416 rad.*

$$\zeta = \sqrt{\frac{\log(M_p)^2}{\pi^2 + \log(M_p)^2}} \tag{3.21}$$

$$\omega_d = \frac{1}{t_r}\left[\pi - \tan^{-1}\left(\frac{\sqrt{1-\zeta^2}}{\zeta}\right)\right] \tag{3.22}$$

$$\omega_n = \frac{\omega_d}{\sqrt{1-\zeta^2}} \tag{3.23}$$

$$M_P[\,\%] = \frac{V_p - V_f}{V_f}(100\,\%) \tag{3.24}$$

Con los parámetros de sobre paso y de sobre elongación ya podemos conocer las constantes propuestas anteriormente sobre la dinámica del motor teniendo como resultado $C_k = 1096.907$ y $C_p = 13.091$. Entonces la Ecuacion 3.25 describe el comportamiento dinámico del motor de CD.

$$\frac{\theta}{\theta_d} = \frac{1096.907}{s(s+13.091)} \tag{3.25}$$

Efectivamente al usar los valores obtenidos mediante la experimentación en la simulación 3.16 y la respuesta transitoria 3.17 en Matlab, se aprecia las mismas características de sobre paso y tiempo de levantamiento.

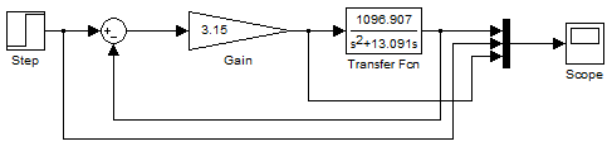


Figura 3.16: *simulación en Matlab del motor con el controlador P.*

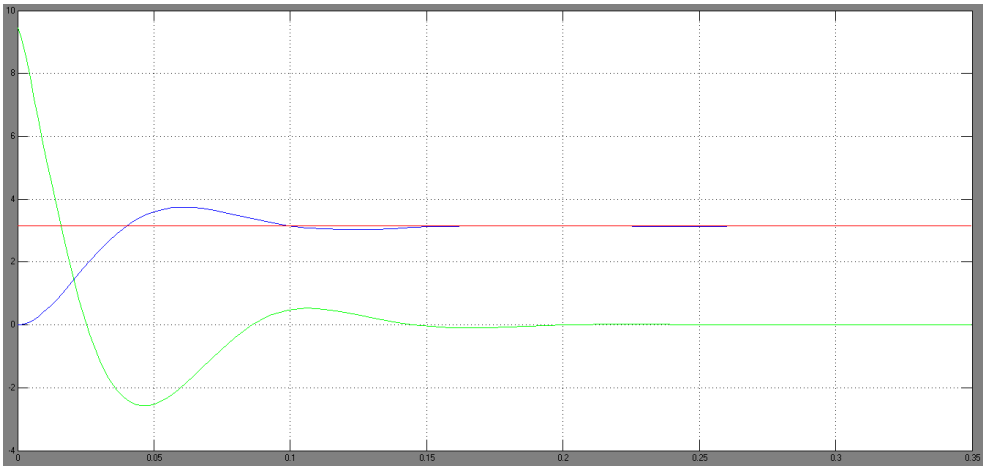


Figura 3.17: *Respuesta transitoria de un escalón al motor con controlador tipo P.*

3.3. Diseño del controlador PD

Un motor como el que se propone aquí en cascada con un controlador PD se puede representar como un sistema de segundo orden también, el cual es ya bastante conocido, también se sabe que para sistemas de dicho orden presentan un factor de amortiguamiento $0 < \zeta < 1$.

Para una entrada tipo escalón unitario se obtiene una respuesta transitoria en estado estable con oscilaciones que se atenúan en un tiempo $t = t_s$ y que debido a esas oscilaciones existe un sobre paso o sobre paso M_p y es fácilmente observable cuando dicha respuesta cruza por primera vez el valor de estado estacionario del sistema en un tiempo t_r.

Se propone un controlador PD para mejorar la respuesta, como se muestra en la Figura 3.18 donde además se quiere que exista un sobrepaso $M_p = 20\,\%$ y y tiempo de levantamiento $t_r = 0.03s$.

Haciendo uso de álgebra de bloques y reduciendo el diagrama mostrado en la

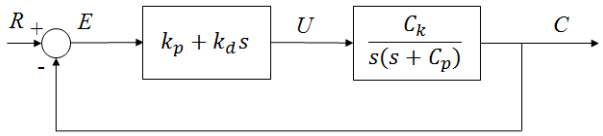


Figura 3.18: *Diagrama de bloques de un controlador y un motor de CD.*

Figura 3.18 se obtiene la expresión 3.26

$$\frac{\theta}{\theta_d} = \frac{sk_dC_k + k_pC_k}{s^2 + s(C_p + k_dC_k) + k_pC_k} \tag{3.26}$$

Entonces utilizando las expresiones 3.21, 3.22, 3.23 y 3.24, donde $\zeta = C_p + k_dC_k$ y $\omega_n^2 = k_pC_k$, puesto a que solo se desconocen k_p y k_d. Entonces se tiene como resultado que $k_p = 3.006$ y $k_d = 0.036$, entonces se simuló en Matlab como se muestra en la Figura 3.19, y se ajustan las ganancias del controlador del motor con los valores ya mencionados y se observa como la respuesta transitoria se comporta de acuerdo a las especificaciones dadas anteriormente de sobre paso y tiempo de respuesta, se puede apreciar esto en la Figura 3.20

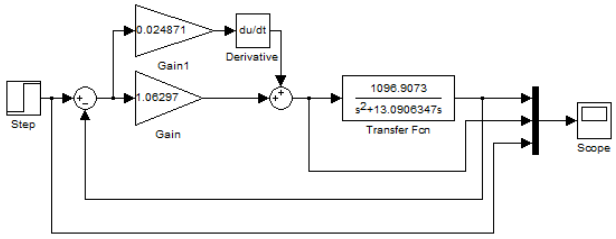


Figura 3.19: *Simulación, controlador realimentado en cascada con un motor.*

La parte Derivativa del controlador ayuda a que el error del sistema sea cero, esto es gracias a que es este quien predice, por así decirlo, el error por lo cual no importa si es mínimo, la naturaleza del sistema lo llevará a cero. Otra razón para esto es que el controlador y el motor forman una Función de Transferencia 3.27 de tipo 1 ya que existe un cero en el origen en el plano complejo y de acuerdo a (Ogata, 2010) un sistema de tipo uno es capaz de seguir escalones y además de alcanzar al escalón, mientras que para una señal tipo rampa solo es capaz de seguir dicha señal pero no alcanzarla, o en otras palabras, habría un error constante diferente de cero para la señal tipo rampa.

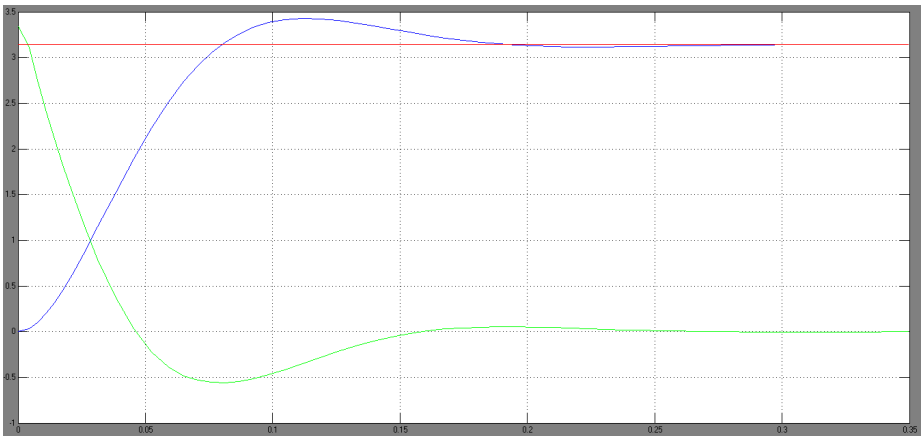


Figura 3.20: *Respuesta del sistema 3.19 para una entrada escalón π.*

$$\frac{E}{C} = \frac{k_p C_K + s k_d C_K}{s(s + C_P)} \tag{3.27}$$

3.4. Descripción de hardware FPGA

En el FPGA se diseño una arquitectura donde se cuentan las cuentas de un motor de CD, dicho encoder es de 448 cuentas por revolución sí se utiliza un precisión de 4x en dicho sensor. El diagrama a bloques de la configuración de la FPGA se muestra en la Figura 3.21. La función principal del FPGA es proveer a la computadora con la información necesaria, en este caso la posición del motor en un momento dado, para que esta pueda efectuar el calculo del controlador y proveer un valor de control en la forma de un duty cycle para controlar el motor mediante un puente H.

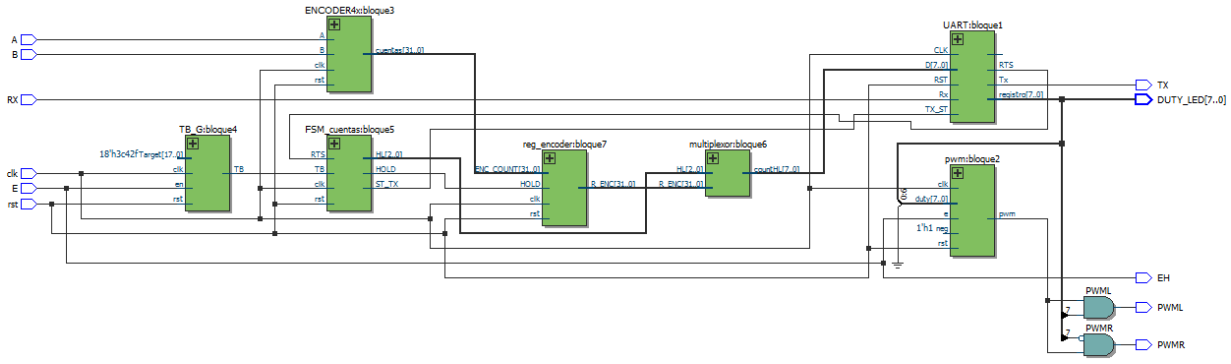


Figura 3.21: *Diagrama de bloques de la configuración de FPGA.*

En esta arquitectura se tiene un contador de 32 bits para contar las cuentas del encoder que pueden ser traducidas a radianes que el rotor recorre en un momento dado. También se encuentran otros dos módulos importantes; RS232 y PWM; estos módulos se encargan de enviar y recibir la información necesaria mediante RS232 y también de establecer un pulso modulado mediante el bloque PWM, con el cual se controla el voltaje aplicado a la armadura del motor.

3.4.1. Decodificador incremental

Este es quien se encarga de identificar la dirección de giro del motor mediante la interpretación de un par de señales en forma de onda cuadrada desfasadas una con respecto de la otra 90deg, tal y como se aprecia en la Figura 3.22. Para poder saber hacia que lado se dirige el motor se identifica un patrón de estas señales para identificar el sentido de giro.

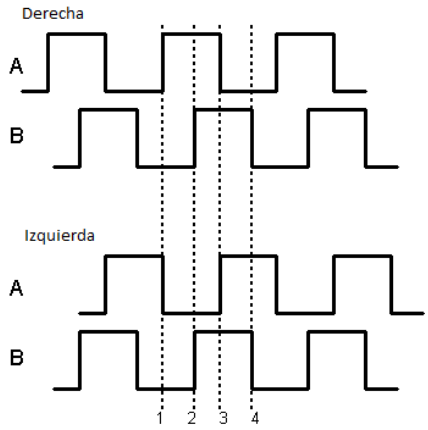


Figura 3.22: *Señales de los canales A y B de un encoder.*

Entonces sí se presenta una secuencia como esta se estaría observando en realidad las señales cuadradas desfasadas en su forma digital:

Primero 00

Segundo 01

Tercero 11

De la misma forma, si estas son leídas de manera inversa se determinara que el motor va en sentido contrario. Para llevar el angulo en unidades de cuentas se lee continuamente estas señales, de este modo, sí se detecta una transición, ya sea hacia arriba o hacia abajo; se decide sí se resta o se incrementa el contador del encoder.

El encoder cuenta con una máquina de estados que se encarga de identificar el sentido de giro y además de incrementar o decrementar la cuenta, dicha cuenta está en complemento a dos o C_2, de este modo se pueden tener cuentas negativas o positivas, el registro donde se aloja este valor tiene una magnitud de 32 bits y se habilita cuando la máquina de estados le ordena mediante la señal 'hold'. Esta decodificador es sincrono y trabaja a la misma velocidad del reloj del FPGA por lo que puede leer máximo una cuenta cada 2 ciclos de reloj, se observa este módulo en la Figura 3.23.

Debido a que las cuentas son distribuidas en números enteros se requiere que se haga una posterior conversión de entero a flotante en la plataforma que se está trabajando, en este caso sería la computadora la que se encargue de realizar esta tarea para no trabajar con cuentas, sino con radianes y así sea más intuitivo el decodificador de posición, como se esta trabajando un un FPGA se puede introducir estos módulos de manera independiente ya que se pueden realizar procesos paralelos en este tipo de plataformas.

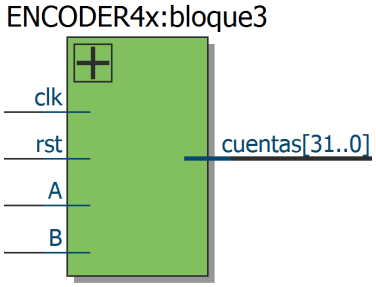


Figura 3.23: *Bloque codificador incremental para el encoder.*

3.4.2. Modulo UART

Se implemento un modulo UART para la transmisión de datos del FPGA a la computadora, este modulo esta dividido en dos partes, la parte de transmisión y la parte de recepción, cada una es independiente de la otra, lo que convierte a este modulo como un transmisor full-duplex. Se manejó este modulo debido a que se cuenta en la tarjeta de desarrollo con un CI(Circuito integrado) MAX232 que se encarga de convertir el protocolo UART en protocolo RS232 que es capaz de manejar los voltajes estandar de $\pm 15V$.

Su forma de opercación es la típica que se usa en la industria; un bit de inicio (0) 8 bits de transmisión de información y un bit de stop, sin paridad ni handshake o coordinación entre la computador y el FPGA.

Como entradas y salidas del módulo UART se tienen las típicas señales de TX y RX, ademas de otras que se pueden usar de manera interna dentro de otro modulo mas grande, como lo es en este proyecto. Se cuentan con otras señales como la bandera de RTS (Ready To Send), o la bandera de dato recibido (INT_ RX) para cuando hay un dato disponible en el puerto que puede ser leído por otro modulo que se encargue solo de estos datos sin interrumpir la transmisión de información del otro lado de dicho modulo.

El puerto UART está configurado para trabajar con una velocidad igual a 115200 baudios, que pues es suficiente para realizar el control en la computadora, pues la recepción de información sería lo suficientemente rápida para realizar control.

Se decidió trabajar bajo este protocolo de comunicación ya que su implementación es bastante rapida, sencilla y fácil de diseñar y adecuar al proyecto, la Figura 3.24 muestra este bloque.

3.4.3. Módulo de PWM

Se construyo un modulo PWM con el cual se controla la alimentación de voltaje del motor mediante el puente H de transistores, Este modulo siempre está activo de acuerdo al byte que se reciba a trabes del puerto serie que en este caso es el registro receptor del

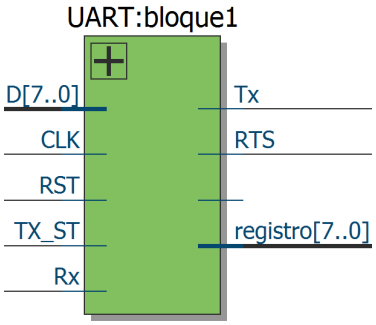


Figura 3.24: *Módulo UART para comunicar con RS232 de la computadora.*

módulo UART. El modulo de PWM, vease 3.26, funciona con una frecuencia de 1KHz. Para que una señal de PWM sea reproducida por el modulo; este debe ser habilitado mediante una señal E. En la Figura 3.25 se observa el modulo PWM, sus entradas y salidas de este.

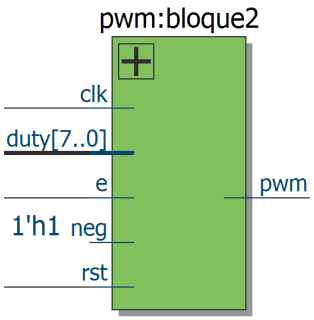


Figura 3.25: *Modulo de PWM.*

Para lograr la inversión del PWM se usa el B8 o el más significativo del dato recibido mediante el puerto RS232, posteriormente, antes de enviar el dato recibido tal cual al modulo PWM se toman posteriormente los bits del 6 al 0 y se concatena un '0' por la derecha o que es lo mismo, se multiplica por 2 y así se recibe el dato al módulo de PWM.

La salida del PWM se puede apreciar en la figura 3.26, este módulo es capaz de invertir la señal de acuerdo a una entrada neg, donde sí esta entrada es '0' no se invierte el PWM, pero si es '1' se invierte.

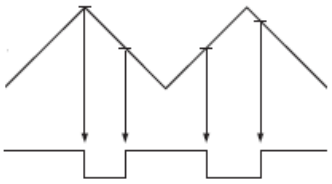


Figura 3.26: *Ciclo de trabajo del PWM.*

3.4.4. Máquina de estados

La máquina de estados es la que se encarga de gestionar todas las tareas del muestreo de la posición del motor ya que gestiona tanto a la parte transmisora del módulo RS232 como un multiplexor y además el registro de las cuentas del encoder.

Este modulo vease la Figura 3.27 comienza su ciclo de trabajo cuando una señal de base de tiempo se activa en '1' indicando el inicio de un ciclo nuevo, esta base de tiempo determina la frecuencia de muestreo realizando un proceso monótono y simple, transfiere 3 bytes mediante el puerto serie a la computadora donde el primer byte es el valor 0x61 el segundo byte es la parte alta de una palabra y el tercer byte es la parte baja de la palabra.

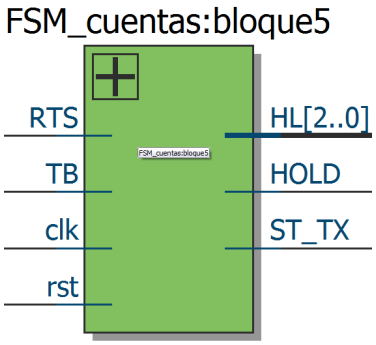


Figura 3.27: *Bloque de la máquina de estados finitos.*

3.4.5. Registro de encoder

Este registro de 32 bits es bastante simple pues solo retiene el dato cuando se le indica mediante una señal que provee la máquina de estados capturando la posición del motor en un momento dado, este registro se muestra en la Figura 3.28.

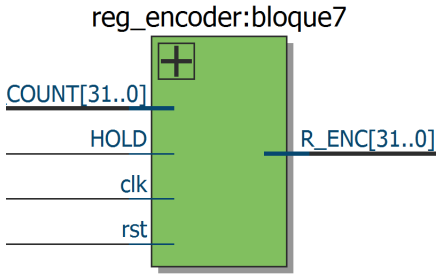


Figura 3.28: *Registro de 32 bits para retención de cuentas del encoder.*

3.4.6. Multiplexor

El multiplexor es bastante simple ya que solo divide la señal del registro de 32 bits en 4 señales más pequeñas de 8 bits cada una y además provee una constante con la cual se inician todas las transmisiones hacia la computadora, estas constante es 0x61, el bloque se puede ver en la Figura 3.29.

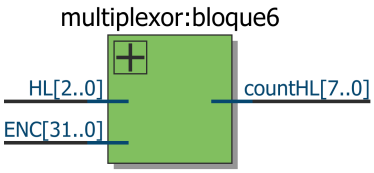


Figura 3.29: *Multiplexor de entrada de 32 bits y salida de 8 bits.*

En este módulo la constante 0x61 se encuentra descrita en el interior de este modulo pues no interesa modificar dicha constante posteriormente.

La frecuencia con que cambia o se mueve el multiplexor depende de la velocidad de transmisión del módulo RS232 pues a mayor baudios la frecuencia de cambió del multiplexor aumenta también ya que este abastece al transmisor del modulo RS232.

3.4.7. Generador de base de tiempo

Este es quien inicia todo, este módulo no es más que un generador genérico de base de tiempo que activa a la máquina de estados cada de que se cumple un tiempo determinado que en este caso es de 5ms, los suficientes como para realizar un control, este módulo como se muestra en la Figura 3.30, es bastante simple y preciso. Debido a sus

características genéricas se puede adecuar para cualquier periodo de muestreo según se requiera. Este modulo precisa de un contador que se incrementa cada 20ns debido al reloj del FPGA de 50Mhz por lo tanto si se quiere mandar un pulso cada .005s entonces dicho contador deberá alcanzar la cuenta de 250000 y hasta entonces se produce un pulso, este pulso es la base de tiempo.

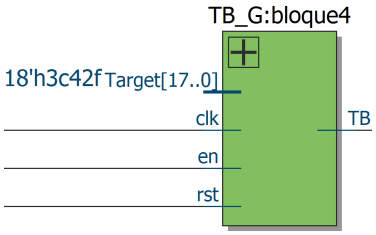


Figura 3.30: *Bloque del generador de base de tiempo.*

3.4.8. Simulación de todo el sistema

Se simuló la configuración del FPGA, en la imagen 3.31 se observa el inicio; poniendo rst a '1', E a '1' consecutivamente e incrementando las cuentas del encoder moviendo a y b consecutivamente una de la otra. En la imagen de la 3.32 se observa que efectivamente que a los 5ms se comienza a enviar el dato de posición actual del motor mediante el el puerto RS232 del FPGA, se envían 3 bytes en esta ocación; byte de inicio, byte de los bits mas altos del contador de posición y por ultimo los bits menos significativos del contador del encoder.

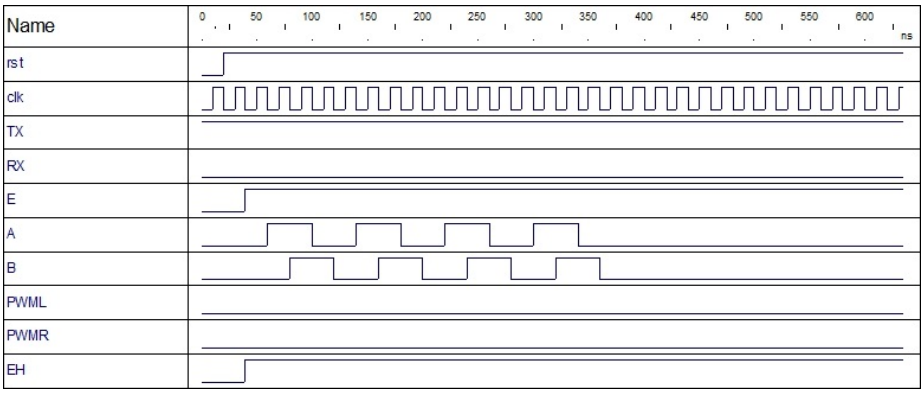


Figura 3.31: *Simulación de código del FPGA de inicio de funcionamiento.*

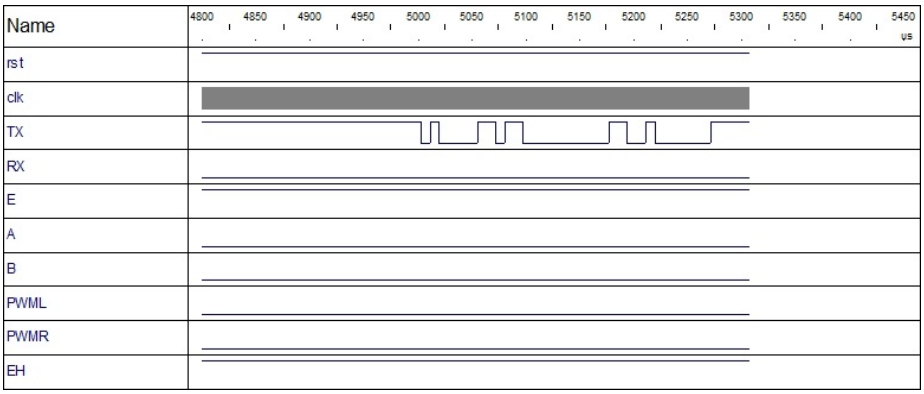


Figura 3.32: *Simulación del código del FPGA del envío de la posición.*

3.5. Controlador

Para el control se optó por utilizar la plataforma de programación builder, en esta plataforma se pueden programar interfaces de usuario de manera sencilla. La plataforma que se usó para el proyecto es Builder 6, en esta se configuró el uso del puerto serial y una interfaz sencilla.

Es en el programa de la computadora donde se realiza el control, es decir, aquí se calcula mediante un error las componentes de un controlador PD pues se multiplica el error y se deriva para posteriormente y es suministrado al motor en forma de voltaje mediante un PWM que se maneja desde la FPGA, en pocas palabras, la computadora es el cerebro de todo mientras que la FPGA actúa como una herramienta de adquisición de datos.

El funcionamiento del controlador es simple, cuando se comienza a recibir información mediante el puerto serial inmediatamente el programa en la computadora comienza a procesar cada byte; sí el byte de inicio es el correcto y los siguientes se comienzan a apilar en variables enteras para almacenar las cuentas recorridas.

En este programa se usa un filtro digital PID de forma continua, es decir, conforme se va muestreando la posición del motor se realiza una integración y derivada.

Para realizar una integral en la computadora se usa un método numérico de integración, en este caso se usa la integral. Entonces, sí se considera como una función del tiempo el error de la suma de la entrada menos la salida se tiene la Ecuación descrita por

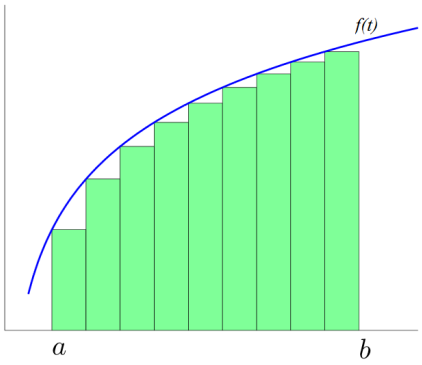


Figura 3.33: *El area bajo la curva se aproxima mediante una suma finita de áreas de rectángulos.*

3.28.

$$e(f) = \sum_{k=0}^{n} = \alpha_k f(x_k) \tag{3.28}$$

Esta técnica calcula el área de columnas o rectángulos que están por debajo de la curva, en este caso la base de dicho rectángulo mide el tiempo de muestreo o lo que es igual 5ms y la altura de este rectángulo es la magnitud del error, puesto a que es una integral también se puede interpretar como un acumulador así que el área del nuevo rectángulo se sumara al resultado total de la integral, tal y como se muestra en la Figura 3.33.

Está claro que cuando no se llega al valor deseado la parte integral sigue acumulando el error hasta que es lo suficientemente grande como para provocar un movimiento en el motor, pero a veces esta parte puede exceder el valor máximo que puede ser producido por la fuente y no importa si este crece no habrá mas voltaje que proveer, pero cuando esta limitación mecánica no existe se recomienda limitar dicha sobre alimentación.

Para el método numérico de la derivada también se deriva la función que describe el error, se usa la derivada hacia atrás como se describe en la Ecuación 3.29. Donde h es el tiempo de 5 ms que es el periodo de muestreo y es en esos instantes donde se tiene un valor de la función del error, véase la Figura 3.34.

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h}{h} \tag{3.29}$$

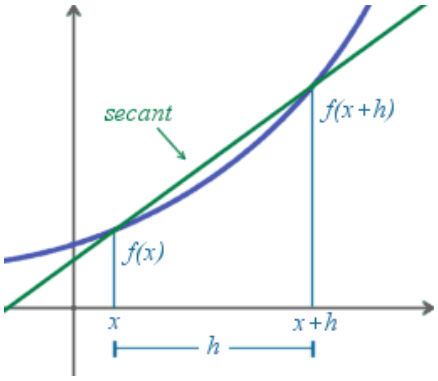


Figura 3.34: Derivada numérica.

La parte proporcional del PID es una simple multiplicación entre una constante y el error, posteriormente el resultado de la parte proporcional mas la parte integral y la parte derivativa devuelve el valor de U que es la entrada de la planta del motor.

Una vez calculado el valor de U mediante el PID se calcula su valor equivalente para un dato de 8 bits puesto que se dispone de un PWM con una resolución de 8 bits y este es enviado a la FPGA. La interfaz de usuario se puede ver en la Figura 3.35.

Figura 3.35: *Interfaz de usuario hecha en Builder.*

Capítulo 4

Resultados y discusión

La mejoría de la respuesta transitoria del motor y además la implementación realizable de un controlador en una computadora y FPGA son tratados en este apartado además de un análisis general los resultados obtenidos.

4.1. Respuesta a un escalón unitario

Para la identificación de la planta se trató de llevar la entrada del motor hasta el nivel mas alto al que se le puede poner, debido a que se uso una fuente conmutada cuyo voltaje nominal es de 11.6 V y las perdidas en el puente H debido a las caídas de tensión en los transistores del mismo, el voltaje alcanza como máximo 10 V aproximadamente, así que para evitar la saturación de la señal de la respuesta transitoria se evito que la ganancia proporcional del controlador superase dicho voltaje.

Se determinaron las ganancias del controlador para mejorar la respuesta transitoria con un sobre paso M_P del 20 % y un tiempo de levantamiento t_r de 0.03 segundos, la respuesta cumple con los parámetros antes mencionados y se puede apreciar en la figura 4.1.

Es evidente la mejoría de la respuesta cuando se incorpora el controlador PID, el sobre paso disminuyó, el tiempo de subida se adecuó al que se había previsto y se mejoro

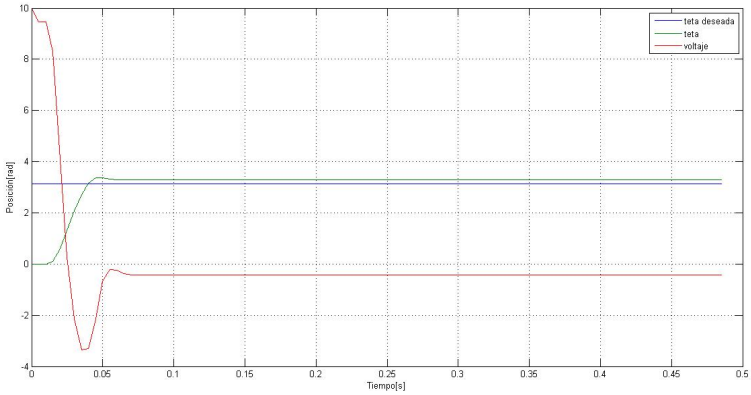


Figura 4.1: *Respuesta a un escalón unitario igual a 3.1416 en el motor.*

el error en estado estacionario, el cual anteriormente era constante y diferente de cero.

Entonces este controlador es capaz de determinar cuando y como se desea la respuesta de un motor, se puede adecuar a cualquier planta, si se necesita mas rapidez o una respuesta más suave, eso ya dependerá de la aplicación en donde se utilice este sistema.

En el aspecto académico este tipo de trabajos pueden proveer una mejor visión para aquellos que incursionen en el estudio de control y puedan ver de primera mano como es que funcionan los controladores, como se desarrolló una interfaz sencilla pero eficaz al momento de realizar control.

4.2. Conclusiones

La experimentación se realizó con éxito, puesto a que solo se alcanzó a desarrollar el controlador de posición esto da pauta para realizar más técnicas de control. No solo se demostró la capacidad de realizar un prototipo computadora-FPGA con un buen rendimiento, sino que también se pudo demostrar como es que el diseño que se realiza en un papel se puede llevar a la realidad, que quizás no se llevó a una implementación de las mas óptimas, si se llevó a una con un desempeño bueno y robusto.

Algunas complicaciones que se tuvieron durante el desarrollo de esta plataforma fue el electrónico puesto que se requiere conectar un circuito que es sensible como un FPGA

y un motor de corriente directa, este actuador produce armónicos o ruido que puede llevar a la parte digital a un rendimiento pobre o equivocado. Se aprovecho la ventaja de que el puente rectificador puede trabajar con casi cualquier nivel de señal digital como en este caso lo fue con 3.3 V, esto redujo la complejidad y permitió una conección casi directa entre el puente H y la tarjeta FPGA.

Las herramientas con las que se diseño el controlador no son complejas o grandes, es decir, las operaciones que se realizan para el PID pueden ser realizadas por un micro controlador y no una computadora, esto tendría como consecuencia un controlador más barato y con una complejidad relativamente mas tediosa en cuanto a la implementación de un MCU, excelente para cuestiones académicas pues se puede implementar cualquier tipo de filtro en un lenguaje de programación de alto nivel que en uno de descripción de hardware.

Por la naturaleza del sistema se cree que es posible realizar toda la tarea de control en el mismo FPGA mediante multiplicadores y acumuladores, puesto a que algunas divisiones se pueden interpretar como multiplicaciones de un número con otro pero inverso, facilitaría más el camino para el desarrollo de una plataforma de este tipo.

Un controlador no siempre funciona con un PWM pero en los servo amplificadores de voltaje de referencia se puede interpretar el duty cycle de un PWM como voltaje de referencia para un DAC de 8 bits por ejemplo y se obtendría un resultado similar.

Bibliografía

Altintas, Y. (2000). *Manufacturing Automation: Metal Cutting Mechanics, Machine Tool Vibrations and CNC Design*. Cambridge Uviversity Press.

Bogdan M. Wiliamowski, J. D. I. (2011). *The industrial electronics Handbook: control and mechatronics*. CRC Press.

Eltra (2000). Sigle electronics s. a.

Firoozian, R. (2009). *Servo Motors and Industrial Control Theory*. Springer.

INEGI, I. N. d. E. y. G. (2011). Sistemas de cuentas nacionales de méxico, producto interno bruto por entidad federativa.

Kariyappa B. S., Hariprasad S. A., R. N. (2009). *Position Control of an AC Servo Motor Using VHDL and FPGA*. World Academy of Science.

Luis F. Castaño Londoño, G. A. O. L. (2011). Diseño de un sistema para el control de posición de un motor dc basado en fpga. Technical report.

Masatoshi Nakamura, Satoru Goto, N. K. (2004). *Mechatronic Servo System Control Problems in Industries and their Theoretical Solutions*. Springer-Verlag.

M.Sivaramakrishna, Chandrakant Upadhyay, C. N. K. (2011). Development of pid controller algorithm over fpga for motor control in failed fuel location module in indian fast reactors. Technical report.

Ángel Colín Robles, J. (2006). Descripción vhdl de los bosques funcionales de un controlador digital pid. Technical report.

Ogata, K. (2010). *Ingeniería de control moderna*. Pearson, 5ta. edición edition.

Osman Koct, Ahmet Teoman Naskali, E. D. K. A. S. (2011). A field programmable gate array based modular motion control platform. Technical report.

savita Sonoli, K. N. R. (2010). Implementation of fpga based pid controller for dc motor speed control system. Technical report.

Wahmad, W. R. B. (2008). A dc motor controller using pid algorithm implementation on pic. Technical report.

Anexo

Código: cont_bis.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity cont_bits is
7    port(
8      RST : in std_logic;
9      CLK : in std_logic;
10     S : in std_logic;
11     INT_RX : in std_logic;
12     CB  : out std_logic_vector(3 downto 0)
13   );
14 end cont_bits;
15
16 architecture Interna of cont_bits is
17 signal Qn, Qp : std_logic_vector(3 downto 0);
18 begin
19   Combinacional : process(Qp, S, INT_RX)
20   begin
21     case INT_RX is
22       when '0' =>
23         if(s = '1')then
24           Qn <= Qp +1;
25         else
```

```
26          Qn <= Qp;
27          end if;
28        when '1' =>
29          Qn <= "0000";
30        when others =>
31          Qn <= "0000";
32      end case;
33      CB <= Qp;
34
35    end process Combinacional;
36
37    Secuencial : process(RST, CLK,Qn)
38    begin
39      if(RST =  '0') then
40        Qp <= (others => '0');
41      elsif(CLK' event and CLK = '1') then
42        Qp <= Qn;
43      end if;
44    end process Secuencial;
45  end Interna;
```

Código: contador_.vhd

```
1  Library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
5
6  Entity Contador_n is
7    Generic(
8      n : integer := 3
9    );
10   Port(
11   rst    : in  std_logic ;
12   clk    : in  std_logic ;
13   target : in  std_logic_vector(n-1 downto 0 ) ;
14   opc    : in  std_logic_vector(1   downto 0 ) ;
```

```
15   eoc    : out std_logic ;
16   eocdown: out std_logic ;
17   cuenta : out std_logic_vector (n-1 downto 0)
18   );
19 end Contador_n;
20
21 Architecture Contador of Contador_n is
22 signal Qn, Qp,cero: std_logic_vector(n-1 downto 0);
23 begin
24   cero <= (others =>'0');
25   Combinacional : process(Qp,opc)
26   begin
27     case opc is
28       when "00" =>
29         Qn <= Qp;
30       when "01" =>
31         Qn <= Qp + 1;
32       when "10" =>
33         Qn <= Qp - 1;
34       when "11" =>
35         Qn <= (others => '0');
36       when others =>
37         Qn <= (others => '0');
38     end case;
39     cuenta <= Qp;
40   end process Combinacional;
41
42   bandera:process(Qp,target)
43   begin
44     if( Qp = target)then
45       eoc <= '1';
46     else
47       eoc <= '0';
48     end if;
49   end process bandera;
50
```

```
  bandera2:process(Qp,cero)
  begin
    if( Qp = cero )then
      eocdown <= '1';
    else
      eocdown <= '0';
    end if;
  end process bandera2;

  Secuencial : process(rst,clk,Qn)
  begin
    if(rst =  '0') then
      Qp <= (others => '0');
    elsif(clk'event and clk = '1') then
      Qp <= Qn;
    end if;
  end process Secuencial;
end Contador;
```

Código: Datos.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Datos is
  port(
  M : in std_logic_vector(3 downto 0);
  D : in std_logic_vector(7 downto 0);
  Tx: out std_logic
  );
end Datos;

architecture Simple of Datos is
begin
  process(M,D)
  begin
    case M is
```

```
      when "0000" => Tx <= '0';
      when "0001" => Tx <= D(0);
      when "0010" => Tx <= D(1);
      when "0011" => Tx <= D(2);
      when "0100" => Tx <= D(3);
      when "0101" => Tx <= D(4);
      when "0110" => Tx <= D(5);
      when "0111" => Tx <= D(6);
      when "1000" => Tx <= D(7);
      when "1001" => Tx <= '1';
      when others => Tx <= '1';
    end case;
  end process;
end Simple;
```

Código: Divisor_833.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Divisor_8333 is
  port(
    RST : in std_logic;
    CLK : in std_logic;
    B : out std_logic
  );
end Divisor_8333;

architecture Interna of Divisor_8333 is
signal Qn, Qp : std_logic_vector(13 downto 0);
begin
  Combinacional : process(Qp)
  begin
    if(Qp = "00000000000000") then
      B <= '1';
```

```
      Qn <= "00000110110010";--9600 -> 01010001010111
                             --115200 -> 00000110110010
    else
      B <= '0';
      Qn <= Qp - 1;
    end if;
  end process Combinacional;

  Secuencial : process(RST, CLK)
  begin
    if(RST =  '0') then
      Qp <= (others => '0');
    elsif(CLK' event and CLK = '1') then
      Qp <= Qn;
    end if;
  end process Secuencial;
end Interna;
```

Código: Divisor_2083.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Divisor_2083 is
  port(
    RST : in std_logic;
    CLK : in std_logic;
    EOR : in std_logic;
    B : out std_logic
  );
end Divisor_2083;

architecture Interna of Divisor_2083 is
signal Qn, Qp : std_logic_vector(11 downto 0);
begin
```

```
18   Combinacional : process(Qp, EOR)
19   begin
20     case EOR is
21       when '0' =>
22         if(Qp = "000000000000") then
23           B <= '1';
24           Qn <= "000011011001";--9600/2 -> 101000101011
25                       --115200/2 -> 000011011001
26         else
27           B <= '0';
28           Qn <= Qp - 1;
29         end if;
30       when others =>
31         Qn <= "000011011001";
32         B <= '0';
33       end case;
34   end process Combinacional;
35
36   Secuencial : process(RST, CLK, Qn)
37   begin
38     if(RST =  '0') then
39       Qp <= (others => '0');
40     elsif(CLK' event and CLK = '1') then
41       Qp <= Qn;
42     end if;
43   end process Secuencial;
44 end Interna;
```

Código: ENCODER4x.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5  entity ENCODER4x is
6    port(
7    rst:    in    std_logic ;
```

```
  clk:    in    std_logic ;
  A:      in    std_logic ;
  B:      in    std_logic ;
  cuentas:  out std_logic_vector (31 downto 0)
  );
end ENCODER4x;
architecture modular of ENCODER4x is
signal Rn,Rp: std_logic_vector(31 downto 0);
signal Qn,Qp: std_logic_vector(3  downto 0);
signal opc  : std_logic_vector(1  downto 0);
begin
  secuencial:process(Qp,A,B)
  begin
    case Qp is
      when "0000" => ----------------S0
        if(A = '0' and B = '0')then
          Qn <= "0000";
        elsif(A = '1' and B = '0')then--goto S1
          Qn <= "0001";
        elsif(A = '0' and B = '1')then--goto S11
          Qn <= "1011";
        end if;
        opc <= "00";--hold

      when "0001" => ----------------S1
        Qn <= "0010";
        opc <= "01";--suma 1

      when "0010" => ----------------S2
        if(A = '1' and B = '0')then
          Qn <= "0010";
        elsif(A = '1' and B = '1')then
          Qn <= "0100";
        elsif(A = '0' and B = '0')then
          Qn <= "0011";
        end if;
```

```
44          opc <= "00";
45
46        when "0011" => ----------------S3
47          Qn <= "0000";
48          opc <= "10";
49
50        when "0100" => ----------------S4
51          Qn <= "0101";
52          opc <= "01";
53
54        when "0101" => ----------------S5
55          if(A = '1' and B = '1')then
56            Qn <= "0101";
57          elsif(A = '0' and B = '1')then
58            Qn <= "0111";
59          elsif(A = '1' and B = '0')then
60            Qn <= "0110";
61          end if;
62          opc <= "00";
63
64        when "0110" => ----------------S6
65          Qn <= "0010";
66          opc <= "10";
67
68        when "0111" => ----------------S7
69          Qn <= "1000";
70          opc <= "01";
71
72        when "1000" => ----------------S8
73          if(A = '0' and B = '1') then
74              Qn <= "1000";
75          elsif(A = '1' and B = '1')then
76            Qn <= "1001";
77          elsif(A = '0' and B = '0')then
78            Qn <= "1010";
79          end if;
```

```
80        opc <= "00";
81
82      when "1001" => ----------------S9
83        Qn <= "0101";
84        opc <= "10";
85
86      when "1010" => ----------------S10
87        Qn <= "0000";
88        opc <= "01";
89
90      when "1011" => ----------------S11
91        Qn <= "1000";
92        opc <= "10";
93
94      when others =>
95      Qn <= (others => '0');
96        opc <= "00";
97    end case;
98  end process secuencial;
99  signAdder:process(Rp,opc)
100 begin
101   case opc is
102     when "00" =>--nada
103       Rn <= Rp;
104     when "01" =>--suma
105       Rn <= Rp + "00000000000000000000000000000001";
106
107     when "10" =>--resta
108       Rn <= Rp + "11111111111111111111111111111111";
109
110     when "11" =>
111       Rn <= Rp;
112
113     when others =>
114       Rn <= Rp;
115   end case;
```

```
116     cuentas <= Rp;
117   end process signAdder;
118   combinacional:process(rst,clk,Qn)
119   begin
120     if(rst = '0')then
121       Qp <= (others => '0');
122       Rp <= (others => '0');
123     elsif(clk'event and clk = '1')then
124       Qp <= Qn;
125       Rp <= Rn;
126     end if;
127   end process combinacional;
128 end modular;
```

Código: FSM_cuentas.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity FSM_cuentas is
5    port(
6    rst: in std_logic ;
7    clk: in std_logic ;
8    TB: in std_logic ;
9    RTS: in std_logic ;
10   ST_TX: out std_logic ;
11   HOLD: out std_logic ;
12   HL: out std_logic_vector (2 downto 0)
13   );
14 end FSM_cuentas;
15
16 architecture FSM of FSM_cuentas is
17 signal Qn, Qp: std_logic_vector (3 downto 0);
18 signal hdlr: std_logic_vector (4 downto 0);
19 begin
20   combinacional:process(TB,Qp,RTS)
21   begin
```

```
22    case Qp is
23      when "0000" =>
24      if(TB='0')then
25        Qn <= Qp;
26      else
27        Qn <= "0001";
28      end if;
29      hdlr<="00000";
30
31      when "0001"=>
32      Qn <= "0010";
33      hdlr<="11000";
34
35      when "0010"=>
36      if(RTS = '0')then
37        Qn <= Qp;
38      else
39        Qn <= "0111";
40      end if;
41      hdlr<="00000";
42
43      when "0011"=>
44      Qn <= "0100";
45      hdlr<="10001";
46
47      when "0100"=>
48      if(RTS='0')then
49        Qn<=Qp;
50      else
51        Qn<="0101";
52      end if;
53      hdlr<="00001";
54
55      when "0101"=>
56      Qn<="0110";
57      hdlr<="10010";
```

```
58
59      when "0110"=>
60      if(RTS='0')then
61        Qn<=Qp;
62      else
63        Qn<="0111";
64      end if;
65      hdlr<="00010";
66
67      when "0111"=>
68      Qn<="1000";
69      hdlr<="10011";
70
71      when "1000"=>
72      if(RTS='0')then
73        Qn<=Qp;
74      else
75        Qn<="1001";
76      end if;
77      hdlr<="00011";
78
79      when "1001"=>
80      Qn<="1010";
81      hdlr<="10100";
82
83      when "1010"=>
84      if(RTS='0')then
85        Qn<=Qp;
86      else
87        Qn<="0000";
88      end if;
89      hdlr<="00100";
90
91      when others =>
92      Qn<="0000";
93      hdlr<="00000";
```

```
94
95      end case;
96    end process combinacional;
97
98    asignacion:process(hdlr)
99    begin
100     ST_TX<=hdlr(4);
101     HOLD<=hdlr(3);
102     HL<=hdlr(2 downto 0);
103   end process asignacion;
104
105   secuencial:process(rst,clk,Qn)
106   begin
107     if(rst='0')then
108       Qp<="0000";
109     elsif(clk'event and clk = '1')then
110       Qp<=Qn;
111     end if;
112   end process secuencial;
113 end FSM;
```

Código: FSM_RX.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_arith.all;
4  use IEEE.std_logic_unsigned.all;
5
6  entity FSM_RX is
7    port(
8    RST : in std_logic;
9    CLK : in std_logic;
10   Rx  : in std_logic; -- bit que da el inicio
11   B   : in std_logic; -- contador
12   CB  : in std_logic_vector(3 downto 0);
13   EOR : out std_logic;
14   S   : out std_logic;
```

```
15   INT_RX: out std_logic
16   );
17 end FSM_RX;
18
19 architecture Control of FSM_RX is
20 signal Qp, Qn : std_logic_vector(3 downto 0);
21 begin
22   combinacional : process(Qp,Rx,B,CB)
23   begin
24     case Qp is
25       when "0000" =>
26         if( Rx = '1') then
27           Qn <= Qp;
28         else
29           Qn <= "0001";
30         end if;
31         EOR <= '1';
32         S   <= '0';
33         INT_RX <= '0';
34       when "0001" =>
35         if( B = '0') then
36           Qn <= Qp;
37         else
38           Qn <= "0010";
39         end if;
40         EOR <= '0';
41         S   <= '0';
42         INT_RX <= '0';
43       when "0010" =>
44         Qn  <= "0011";
45         EOR <= '0';
46         S   <= '1';
47         INT_RX <= '0';
48       when "0011" =>
49         if(B = '0') then
50           Qn <= Qp;
```

```
51          else
52            if(CB = "1010")then
53              Qn <= "0100";
54            else
55              Qn <= "0001";
56            end if;
57          end if;
58          EOR <= '0';
59          S   <= '0';
60          INT_RX <= '0';
61        when "0100" =>
62          Qn <= "0000";
63          EOR <= '1';
64          S <= '0';
65          INT_RX <= '1';
66        when others =>
67          Qn <= "0000";
68          EOR <= '1';
69          S   <= '0';
70          INT_RX <= '0';
71        end case;
72      end process Combinacional;
73
74      Secuencial : process(Qn,RST, CLK)
75      begin
76        if(RST = '0') then
77          Qp <= (others => '0');
78        elsif(CLK' event and CLK = '1') then
79          Qp <= Qn;
80        end if;
81      end process Secuencial;
82  end Control;
```

Código: FSM_TX.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
```

```
3
4  entity FSM_TX is
5    port(
6      RST : in std_logic;
7      CLK : in std_logic;
8      B : in std_logic;
9      STT : in std_logic;
10     RTS : out std_logic;
11     M   : out std_logic_vector(3 downto 0)
12   );
13 end FSM_TX;
14
15 architecture Control of FSM_TX is
16 signal Qn, Qp : std_logic_vector(3 downto 0);
17 begin
18   Combinacional : process(Qp, STT, B)
19   begin
20     case Qp is
21       when "0000" =>        -- Estado 0
22         if(STT = '0') then
23           Qn <= "0000"; --mantener
24         else
25           Qn <= "0001";
26         end if;
27         RTS <= '1';
28         M <= "1111";
29       when "0001" =>       -- Estado 1
30         if(B = '0') then
31           Qn <= Qp;
32         else
33           Qn <= "0010";
34         end if;
35         RTS <= '0';
36         M <= "1111";
37       when "0010" =>       -- Estado 2
38         if(B = '0') then
```

```
39          Qn <= Qp;
40        else
41          Qn <= "0011";
42        end if;
43        RTS <= '0';
44        M <= "0000";    -- Start
45      when "0011" =>      -- Estado 3
46        if(B = '0') then
47          Qn <= Qp;
48        else
49          Qn <= "0100";
50        end if;
51        RTS <= '0';
52        M <= "0001";   -- Dato D(0)
53      when "0100" =>     -- Estado 4
54        if(B = '0') then
55          Qn <= Qp;
56        else
57          Qn <= "0101";
58        end if;
59        RTS <= '0';
60        M <= "0010";   -- Dato D(1)
61      when "0101" =>     -- Estado 5
62        if(B = '0') then
63          Qn <= Qp;
64        else
65          Qn <= "0110";
66        end if;
67        RTS <= '0';
68        M <= "0011";  -- Dato D(2)
69      when "0110" =>    -- Estado 6
70        if(B = '0') then
71          Qn <= Qp;
72        else
73          Qn <= "0111";
74        end if;
```

```
75          RTS <= '0';
76          M <= "0100";  -- Dato D(3)
77        when "0111" =>    -- Estado 7
78          if(B = '0') then
79            Qn <= Qp;
80          else
81            Qn <= "1000";
82          end if;
83          RTS <= '0';
84          M <= "0101";  -- Dato D(4)
85        when "1000" =>    -- Estado 8
86          if(B = '0') then
87            Qn <= Qp;
88          else
89            Qn <= "1001";
90          end if;
91          RTS <= '0';
92          M <= "0110";  -- Dato D(5)
93        when "1001" =>    -- Estado 9
94          if(B = '0') then
95            Qn <= Qp;
96          else
97            Qn <= "1010";
98          end if;
99          RTS <= '0';
100         M <= "0111";  -- Dato D(6)
101       when "1010" =>    -- Estado 10
102         if(B = '0') then
103           Qn <= Qp;
104         else
105           Qn <= "1011";
106         end if;
107         RTS <= '0';
108         M <= "1000";  -- Dato D(7)
109       when "1011" =>    -- Estado 11
110         if(B = '0') then
```

```
111           Qn <= Qp;
112         else
113           Qn <= "0000";
114         end if;
115         RTS <= '0';
116         M <= "1001";     -- Parity
117       when others =>     -- Estado 12
118         Qn <= "0000";
119         RTS <= '0';
120         M <= "1111";
121     end case;
122   end process Combinacional;
123
124   Secuencial : process(RST, CLK)
125   begin
126     if(RST =  '0') then
127       Qp <= (others => '0');
128     elsif(CLK' event and CLK = '1') then
129       Qp <= Qn;
130     end if;
131   end process Secuencial;
132 end Control;
```

Código: latch.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity latchR is
5    port(
6    rst: in std_logic ;
7    clk: in std_logic ;
8    D: in std_logic ;
9    Q: out std_logic
10   );
11 end latchR;
12 architecture enclave of latchR is
```

```
signal Qp,Qn:std_logic ;
begin
  Q<= Qp;

  combinacional:process(Qp,D)
  begin
    if(D = '1')then
      if(Qp = '1')then
        Qn <= '0';
      else
        Qn <= '1';
      end if;
    else
      Qn <= Qp;
    end if;
  end process combinacional;

  secuencial:process(clk,rst,Qn)
  begin
    if(rst = '0')then
      Qp <= '0';
    elsif(clk'event and clk = '1')then
      Qp <= Qn;
    end if;
  end process secuencial;
end enclave;
```

Código: monitorposision.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity monitorposision is
  port(
  rst: in std_logic ;
  clk: in std_logic ;
```

```
  TX: out std_logic ;
  RX: in std_logic ;
  E: in std_logic ;
  A: in std_logic ;
  B: in std_logic ;
  PWML: out std_logic ;
  PWMR: out std_logic ;
  DUTY_LED: out std_logic_vector(7 downto 0);
  EH: out std_logic
  );
end monitorposision;

architecture top of monitorposision is
component UART
  port(
    RST     : in  std_logic;
    CLK     : in  std_logic;
    TX_ST     : in  std_logic;
    D       : in  std_logic_vector(7 downto 0);
    RTS     : out   std_logic;
    Tx      : out   std_logic;
    Rx      : in  std_logic;
    registro  : out   std_logic_vector(7 downto 0);
      INT_RX    : out std_logic
  );
end component ;
component pwm
  port(
  rst: in std_logic ;
  clk: in std_logic ;
  e: in std_logic ;
  duty : in std_logic_vector (7 downto 0);
  neg : in std_logic ;
  pwm : out std_logic
  );
end component;
```

```
45
46 component ENCODER4x
47   port(
48   rst:    in    std_logic ;
49   clk:    in    std_logic ;
50   A:      in    std_logic ;
51   B:      in    std_logic ;
52   cuentas:  out   std_logic_vector (31 downto 0)
53   );
54 end component;
55
56 component TB_G
57   generic(
58   n : integer := 5
59   );
60   port(
61   rst : in  std_logic ;
62   clk : in  std_logic ;
63   en  : in  std_logic ;
64   Target: in std_logic_vector (n-1 downto 0) ;
65   TB  : out std_logic
66   );
67 end component;
68
69 component FSM_cuentas
70   port(
71   rst: in std_logic ;
72   clk: in std_logic ;
73   TB: in std_logic ;
74   RTS: in std_logic ;
75   ST_TX: out std_logic ;
76   HOLD: out std_logic ;
77   HL: out std_logic_vector (2 downto 0)
78   );
79 end component;
80 component multiplexor
```

```
  port(
  R_ENC:  in std_logic_vector(31 downto 0);
  HL:     in std_logic_vector (2 downto 0);
  countHL: out std_logic_vector(7 downto 0)
  );
end component;
component reg_encoder
  port(
  rst: in std_logic;
  clk: in std_logic;
  ENC_COUNT: in std_logic_vector(31 downto 0);
  HOLD : in std_logic;
  R_ENC: out std_logic_vector(31 downto 0)
  );
end component;

signal LR,ST_TX,RTS,TB,HOLD,RX_INT,PWMs: std_logic ;
signal HL:std_logic_vector (2 downto 0);
signal ENC_COUNT,R_ENC:std_logic_vector (31 downto 0);
signal countHL,DUTY,RX_REG:std_logic_vector (7 downto 0);
begin
  LR<=RX_REG(7);
  EH<=E;
  PWML<=PWMs AND LR;
  PWMR<=PWMs AND (NOT LR);
  DUTY_LED<=RX_REG;

  corrimiento:process(RX_REG)
  begin
    DUTY <= RX_REG(6 downto 0) & '0';
  end process corrimiento;



  bloque1: UART port map(rst,clk,ST_TX,countHL,RTS,TX,RX,RX_REG,RX_INT);
  bloque2: PWM  port map(rst,clk,E,DUTY,'1',PWMs);
```

```
117   bloque3: ENCODER4x port map(rst,clk,A,B,ENC_COUNT);
118   bloque4: TB_G generic map (18)port map(rst,clk,E,"1111010000010001111",TB);
       --111101000010001111
119   bloque5: FSM_cuentas port map(rst,clk,TB,RTS,ST_TX,HOLD,HL);
120   bloque6: multiplexor port map(R_ENC,HL,countHL);
121   bloque7: reg_encoder port map(rst,clk,ENC_COUNT,HOLD,R_ENC);
122 end top;
```

Código: mux.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity multiplexor is
6    port(
7    R_ENC:  in std_logic_vector(31 downto 0);
8    HL:     in std_logic_vector (2 downto 0);
9    countHL: out std_logic_vector(7 downto 0)
10   );
11 end multiplexor;
12 architecture algo of multiplexor is
13 begin
14   muxp:process(R_ENC,HL)
15     begin
16       case HL is
17         when "000" => countHL <= "01100001";
18         when "001" => countHL <= R_ENC(31 downto 24);
19         when "010" => countHL <= R_ENC(23 downto 16);
20         when "011" => countHL <= R_ENC(15 downto 8);
21         when "100" => countHL <= R_ENC(7  downto 0);
22         when others=> countHL <= "01100001";
23       end case;
24   end process muxp;
25 end algo;
```

Código: Receptor_RS232.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Receptor_RS232 is
  port(
  RST     : in  std_logic;
  CLK     : in  std_logic;
  Rx      : in  std_logic;
  registro  : out std_logic_vector(7 downto 0);
  INT_RX    : out std_logic
  );
end Receptor_RS232;

architecture Bloques of Receptor_RS232 is

component FSM_RX
  port(
  RST : in std_logic;
  CLK : in std_logic;
  Rx  : in std_logic;
  B   : in std_logic;
  CB  : in std_logic_vector(3 downto 0);
  EOR : out std_logic;
  S   : out std_logic;
  INT_RX: out std_logic
  );
end component;

component Divisor_2083
  port(
    RST : in std_logic;
    CLK : in std_logic;
    EOR : in std_logic;
    B : out std_logic
  );
end component;
```

```
37
38 component Registro_D
39   port(
40   RST : in std_logic;
41   CLK : in std_logic;
42   Rx  : in std_logic;
43   S   : in std_logic;
44   R   : out std_logic_vector(7 downto 0)
45   );
46 end component;
47
48 component cont_bits
49   port(
50     RST : in std_logic;
51     CLK : in std_logic;
52     S : in std_logic;
53     INT_RX : in std_logic;
54     CB  : out std_logic_vector(3 downto 0)
55   );
56 end component;
57
58 signal R : std_logic_vector(7 downto 0); --bytes del registro, debo mapear
     este hasta la salida del rs-232  (r)
59 signal CB : std_logic_vector(3 downto 0);
60 Signal B, S, EOR, INT_RX_s : std_logic;
61
62 begin
63   Modulo_1 : FSM_RX         port map (RST, CLK, Rx, B, CB, EOR,S,INT_RX_s);
64   Modulo_2 : Divisor_2083   port map (RST, CLK, EOR, B);
65   Modulo_3 : Registro_D     port map (RST, CLK, Rx, S, R);
66   Modulo_5 : cont_bits      port map (RST, CLK, S, INT_RX_s, CB);
67   registro <= R;
68   INT_RX   <= INT_RX_s;
69 end Bloques;
```

Código: reg_encoder.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity reg_encoder is
  port(
  rst: in std_logic;
  clk: in std_logic;
  ENC_COUNT: in std_logic_vector(31 downto 0);
  HOLD : in std_logic;
  R_ENC: out std_logic_vector(31 downto 0)
  );
end reg_encoder;

architecture registro of reg_encoder is
signal Qn,Qp: std_logic_vector ( 31 downto  0);
begin
combinacional:process(Qp,ENC_COUNT,HOLD)
  begin
    if(HOLD='1')then
      Qn<=ENC_COUNT;
    else
      Qn<=Qp;
    end if;
  end process combinacional;
  R_ENC <= Qp;
  secuencial:process(rst,clk,Qn)
  begin
    if(rst='0')then
      Qp<=(others=>'0');
    elsif(clk'event and clk = '1') then
      Qp<=Qn;
    end if;
  end process secuencial;
end registro;
```

Código: Registro_D.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Registro_D is
  port(
  RST : in std_logic;
  CLK : in std_logic;
  Rx  : in std_logic;
  S   : in std_logic;
  R   : out std_logic_vector(7 downto 0)
  );
end Registro_D;

architecture Derecha of Registro_D is
signal Qn, Qp : std_logic_vector(8 downto 0);
begin
  Combinacional : process(Qp, Rx, S)
  begin
    if(S = '0') then
      Qn <= Qp;
    else
      Qn <= Rx & Qp(8 downto 1); -- a la derecha
    end if;
    R <= Qp(7 downto 0);
  end process Combinacional;

  Secuencial : process(RST, CLK, Qn)
  begin
    if(RST = '0') then
      Qp <= (others => '0');
    elsif(CLK' event and CLK = '1') then
      Qp <= Qn;
    end if;
  end process Secuencial;
end Derecha;
```

Código: TB_G.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity TB_G is
6    generic(
7    n : integer := 5
8    );
9    port(
10   rst : in  std_logic ;
11   clk : in  std_logic ;
12   en  : in  std_logic ;
13   Target: in std_logic_vector (n-1 downto 0) ;
14   TB  : out std_logic
15   );
16 end TB_G;
17
18 architecture tiempo of TB_G is
19 signal Qp,Qn : std_logic_vector ( n-1 downto 0) ;
20 begin
21   combinacional:process(Qp,en,Target)
22   begin
23     if(en = '1') then
24       if(Qp = Target) then
25         TB <= '1';
26         Qn <= (others => '0');
27       else
28         TB <= '0';
29         Qn <= Qp + 1 ;
30       end if;
31     else
32       TB <= '0';
33       Qn <= (others => '0');
34     end if;
35   end process combinacional;
```

```
36
37   secuencial:process(rst,clk,Qn)
38   begin
39     if(rst = '0') then
40       Qp <= (others => '0');
41     elsif(clk'event and clk = '1')then
42       Qp <= Qn;
43     end if;
44   end process secuencial;
45 end tiempo;
```

Código: top_pwm.vhd

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4
5  entity pwm is
6    port(
7    rst: in std_logic ;
8    clk: in std_logic ;
9    e: in std_logic ;
10   duty : in std_logic_vector (7 downto 0);
11   neg : in std_logic ;
12   pwm : out std_logic
13   );
14 end pwm;
15
16 architecture top of pwm is
17
18 component Contador_n
19   Generic(
20     n : integer := 3
21   );
22   Port(
23   rst    : in  std_logic ;
24   clk    : in  std_logic ;
```

```
  target : in  std_logic_vector(n-1 downto 0 ) ;
  opc    : in  std_logic_vector(1   downto 0 ) ;
  eoc    : out std_logic ;
  eocdown: out std_logic ;
  cuenta : out std_logic_vector (n-1 downto 0)
  );
end component;

component latchR
  port(
  rst: in std_logic ;
  clk: in std_logic ;
  D: in std_logic ;
  Q: out std_logic
  );
end component;

component TB_G
  generic(
  n : integer := 5
  );
  port(
  rst : in  std_logic ;
  clk : in  std_logic ;
  en  : in  std_logic ;
  Target: in std_logic_vector (n-1 downto 0) ;
  TB  : out std_logic
  );
end component;

signal opc: std_logic_vector (1 downto 0);
signal eoc,eocdown, eocounts ,TB,c1,c2,L: std_logic;
signal count: std_logic_vector (7 downto 0);
begin

```

```
60   bloque1: Contador_n generic map(8) port map(rst,clk,"11111111",opc,eoc,
      eocdown,count);
61   bloque2: latchR port map(rst,tb,eocounts,L);
62   bloque3: TB_G generic map(5) port map(rst,clk,E,"00011",TB);
63
64   OPC <= ( NOT(E) OR (L AND TB) ) & ( NOT(E) OR ( (NOT L) AND TB ));
65   eocounts <= eoc OR (eocdown AND L );
66
67   mux:process(c1,c2,neg)
68   begin
69     if (neg = '0') then
70       pwm <= c1;
71     else
72       pwm <= c2;
73     end if;
74   end process mux;
75
76   comparador:process(count, duty)
77   begin
78     if(count > duty)then
79       c1 <= '1';
80     else
81       c1 <= '0';
82     end if;
83
84     if(count < duty)then
85       c2 <= '1';
86     else
87       c2 <= '0';
88     end if;
89   end process comparador;
90 end top;
```

Código: Transmisor_RS232.vhd

```
1 library IEEE;
2 use IEEE.std_logic_1164.all;
```

```
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Transmisor_RS232 is
  port(
    RST : in std_logic;
    CLK : in std_logic;
    TX_ST : in std_logic;
    D   : in std_logic_vector(7 downto 0);
    RTS : out std_logic;
    Tx  : out std_logic
  );
end Transmisor_RS232;

architecture Simple of Transmisor_RS232 is

component FSM_TX
  port(
    RST : in std_logic;
    CLK : in std_logic;
    B : in std_logic;
    STT : in std_logic;
    RTS : out std_logic;
    M   : out std_logic_vector(3 downto 0)
  );
end component;

component Divisor_8333
  port(
    RST : in std_logic;
    CLK : in std_logic;
    B : out std_logic
  );
end component;

component Datos
```

```
  port(
  M : in std_logic_vector(3 downto 0);
  D : in std_logic_vector(7 downto 0);
  Tx: out std_logic
  );
end component;

signal Es: std_logic_vector(3 downto 0);
signal B : std_logic;
begin
  Data  : Datos       port map(Es,D,Tx);
  FSM   : FSM_TX      port map(RST, CLK,B,TX_ST,RTS,Es);
  TB    : Divisor_8333  port map(RST,CLK,B);
end Simple;
```

Código: UART.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity UART is
  port(
    RST     : in  std_logic;
    CLK     : in  std_logic;
    TX_ST     : in  std_logic;
    D       : in  std_logic_vector(7 downto 0);
    RTS     : out   std_logic;
    Tx      : out   std_logic;
    Rx      : in  std_logic;
    registro  : out   std_logic_vector(7 downto 0);
      INT_RX    : out std_logic
  );
end UART;

architecture Simple of UART is
```

```
21
22 component Transmisor_RS232
23   port(
24     RST : in std_logic;
25     CLK : in std_logic;
26     TX_ST : in std_logic;
27     D   : in std_logic_vector(7 downto 0);
28     RTS : out std_logic;
29     Tx  : out std_logic
30   );
31 end component;
32
33 component Receptor_RS232
34   port(
35   RST     : in  std_logic;
36   CLK     : in  std_logic;
37   Rx      : in  std_logic;
38   registro  : out std_logic_vector(7 downto 0);
39   INT_RX    : out std_logic
40   );
41 end component;
42   signal R: std_logic_vector(7 downto 0);
43 begin
44   Modulo_env : Transmisor_RS232 port map(RST,CLK,TX_ST,D,RTS,Tx);
45   Modulo_rec : Receptor_RS232   port map(RST, CLK, Rx, R,INT_RX);
46   registro <= R;
47 end simple;
```